VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SIMULÁTOR LABORATORNÍHO MODELU
SIMULATOR OF PHYSICAL MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Štěpán Šváb**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Václav Kaczmarczyk, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Štěpán Šváb ***ID:*** 173762
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Simulátor laboratorního modelu

POKYNY PRO VYPRACOVÁNÍ:

1) Popište základy renderování objektů na scéně – zaměřte se na vysvětlení funkce jednotlivých matic a matematických operací s nimi prováděných.
2) Vytvořte parser X3D formátu v jazyce C#, který naplní data pro svět geometrie, svět dynamiky a klouby v enginu fyziky - BulletSharp.
3) Vytvořte renderer základních geometrických tvarů ze scény z enginu BulletSharp.
4) Vytvořte a otestujte simulátor používající vybraný engine fyziky a vytvořený renderer ovládaný pomocí standardních prvků windows formuláře.
5) Otestujte simulátor na demonstračním modelu.

DOPORUČENÁ LITERATURA:

Chris Dickinson: Learning Game Physics with Bullet Physics and OpenGL

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 29.5.2017

Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá základními metodami vyrenderování scény. Tvorbou simulátoru laboratorního modelu v jazyce C# s využitím bullet phyzic, která definuje svět fyziky. Vytvořením parseru X3D formátu a způsobem, jak jeho daty naplnit svět fyziky.

KLÍČOVÁ SLOVA

2D, 3D, GPU, CPU, OpenGL, pipelining, vertex, pixel, shared, scéna, transformation, simulátor, objekt, tag, element, parser, RigidBody, bullet physics, shape, .NET framework, windows formulář, testování, FPS

ABSTRACT

The aim of this thesis is basic methods of rendering the scene. By creating a simulation model in C# using the bullet phyzic that defines the world of physics. Creating an parser of X3D format and how its data will fill the world of physics.

KEYWORDS

2D, 3D, GPU, CPU, OpenGL, pipelining, vertex, pixel, shared, scene, transformation, simulator, object, tag, element, parser, RigidBody, bullet physics, shape, .NET framework, windows form, testing, FPS

ŠVÁB, Štěpán *Vizualizace pro simulátor laboratorního modelu*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky, 2017. 58 s. Vedoucí práce byl Ing. Václav Kaczmarczyk, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vizualizace pro simulátor laboratorního modelu“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Václavu Kaczmarczykovi Ph.D. za odborné vedení, konzultace, trpělivost.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Počítačová grafika je obor, který je využíván a vyvíjen už od počátku prvních počítačů. Tento obor se zabývá mnoha směry. Jedním z těchto oborů je i tvorba simulátoru. Ať už pro urychlení vývoje nebo pro zjednodušení práce či vyladění chyb nějakého reálného přístroje. Tato práce je zaměřena především na vytvoření simulátoru laboratorního modelu v jazyce C# s využitím Bullet phyzic, která definuje svět fyziky.

V první části tohoto projektu je popsán pojem rendrování a popis, jak se vyrendruje základní scéna. Také jsou zde vysvětleny základní transformace, které lze s jakýmkoli objektem ve scéně provézt.

V druhé části je popsán formát X3D, z čeho je tvořen a jaká jsou jeho základní sintaktická pravidla.

Třetí část popisuje knihovnu bullet phyzic a jak za pomocí této knihovny vytvořit tuhá tělesa a jak nadále s nimi pracovat. Je tady také popis vytváření jednotlivých primitiv, omezení jejich pohybu a spojení těchto objektů mocí kloubů. Posléze je vysvětlen raytest a jak se uplatní síly motoru na tělesa ve scéně.

Čtvrtá část se již věnuje praktické části mojí práce vytvoření parseru, který vybere data z X3D, které popisuje scénu a tato data následně vykreslí do scény.

V páté části práce byl vytvořen laboratorní model kmeny. Tento model byl následně v kapitole šest otestován za jakých podmínek může fungovat.

1 ZÁKLADY RENDEROVÁNÍ OBJEKTŮ NA SCÉNĚ

Renderování je tvorba reálného obrazu na základě PC modelu. Je závislá na několika parametrech, kterými lze obraz (scénu) ovlivňovat. Rendrovací nebo obrazová syntéza je proces vykreslování obrazu 2D nebo 3D modelů do scény pomocí počítačových programů. Také výsledky tohoto modelu leze nazvat rendering. Soubor scéna obsahuje objekty v přesně vymezené jazykové nebo datové struktuře, to může zahrnovat geometrii, viewpoint (místo odkud se na objekt díváme), textury, osvětlení a stínování, což jsou informace popisující virtuální scénu. Údaje obsažené v souboru scény jsou pak předány do asanačního programu, který má být zpracován jako výstup do digitálního obrazu nebo registrové grafiky souboru obrazu. Pod pojmem "rendering"může být analogií uměleckého ztvárnění scény. Ačkoli technické podrobnosti vykreslovacích metod se liší obecnými problémy při výrobě 2D obrazu z 3D. Postup zobrazení je uložený v souboru scény jako grafické potrubí(pipelining), které se aplikuje přímo na GPU. Grafickou procesorovou jednotkou je účelová zařízení schopné pomáhat CPU při provádění složitých výpočtů vykreslování. Pokud by měla scéna vypadat poměrně realisticky a předvídatelně v rámci virtuálního osvětlení, rendering software by měl řešit zobrazovací rovnici(rendering equation). Vykreslovací rovnice nepočítá všechny světelné jevy, ale je obecný osvětlovací model pro počítačově generované snímky. Rendering je také používán k popisu procesu výpočtu efektů v programu pro úpravy videa a dokáže produkovat finální video výstupu. Rendering je jednou z hlavních dílčích témat 3D počítačové grafiky, v praxi je vždy spojena s ostatními. V grafickém potrubí (pipelining) je posledním a velice důležitým krokem konečný vzhled modelů a animací. Se zvyšující se složitostí počítačové grafiky od roku 1970 se stává rendering výrazným předmětem grafiky. Rendering má využití v architektuře, videohrách, simulátorech, filmech nebo televizní vizualizaci efektů. Vizualizace návrhů, z nichž každý zaměstnává jinou rovnováhu mezi funkcí a technikou. Existuje široká škála rendererů, které jsou k dispozici. Některé z nich jsou integrovány do větších modelovacích a animačních balíčků, některé jsou samostatné (stand-alone), některé jsou zdarma (open-source). Na vnitřní straně renderer je pečlivě navržený program založený na selektivních oblastech souvisejících se světelnou fyzikou, zrakovým vnímáním, matematikou a vývojem softwaru. V případě 3D grafiky vizualizace může být provedena pomalu, buď v pre-renderingu (vykreslení dopředu a pote jen zobrazeno), nebo v reálném čase. Pre-rendering je výpočetně intenzivní proces, který se obvykle používá pro tvorbu filmu, zatímco v reálném čase je renderování často děláno pro 3D videohry, které se opírají o využití grafických karet s 3D hardwarovými akcelerátory. [1] [2]

1.1 Pipeline

V dnešní době jsou zobrazovací pipeline velice složité a propracované programy, které umožnují rychlou a efektivní práci s daty. Názornou ukázku pipeline OpenGl 4.4 najdeme na stránkách [1] . Tyto pipeline mají několik důležitých částí, díky kterým můžeme zobrazovat jednotlivé objekty na scéně. Nejdůležitější jsou vertex shader a fragment shader. [11] [5]

1.1.1 Vertex shader

Vertex shader stage zpracovává a vytváří vrcholy ze vstupního asembler kódu, s kterými je pak schopen provádět např. transformace a osvětlení vrcholu. Vestex shader je schopen však v jeden okamžik pracovat jen s jedním vstupem, ze kterého vytvoří vrchol. Shader je vždy aktivní, pokud je vrchol vykreslován. Pokud není zapotřebí žádná změna nebo transformace vrcholu, uloží se do pipline(potrubí) vykreslovacího řetězce. Díky tomuto shaderu můžeme provádět jednotlivé transformace které si vysvětlíme později.

1.1.2 Pixel shader

Pixel shader, nebo také někdy označován fragment shader, je schopen přepočítávat funkci pixel po pixelu. Díky němu můžeme zobrazovat ohromné množství materiálů, jasů, efektů a měnit barvu a hloubku. Také umožňuje velkou škálu stínící techniky (per-pixel,osvětlení). Můžeme tedy říct, že Pixel shader je přímo ten, co ovlivňuje výstup na obrazovku. Vstupními daty pro Pixel shader jsou atributy jednotlivých vertexů, které mohou být považovány za primitivní konstanty. [3]

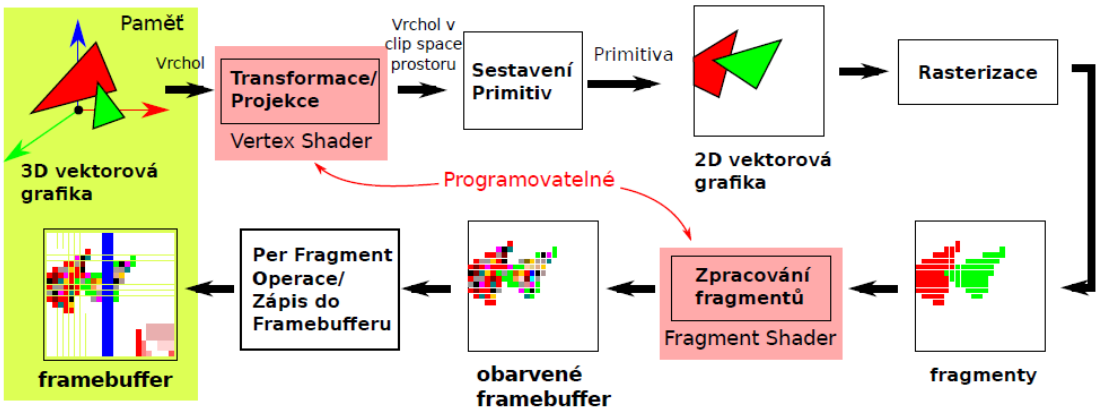
1.1.3 Další druhy shaderů

Postupem času vznikají nové shadery, díky kterým je možné lépe pracovat s daty. V roce 2009 vznikl Direct3D 11 a OpenGl 3.2, které rozšířili vykreslovací řetězce o metodu teselace. Directx (Hull shader, domain shader), OpenGL (Tesselation control shader, teselation evolution shader). Díky umístění teselace přímo na desky grafických karet nebo do čipové sady je možné přidat do scény velké množství detailů V roce 2012 vznikl Direct3D 11.1 a OpenGl 4.3 Compute shadingy, který využívá paralelních procesů přímo na grafické kartě k výpočtu obecných algoritmů (flexibilní vykreslování řetězců). Dříve se používalo fixní vykreslování grafických řetězců.

[1] `https://openglinsights.com/pipeline.html`

Nakonec je také důležité zmínit, že v dnešní době jsou tyto shadery programovatelné a díky tomu umožnují zkušeným programátorům zrychlovat vykreslovací výkon pipeliny pro jenich scénu [4].

Pro názorné zobrazení jak může primitivní pipeline s Fragment shaderem a Vertex shaderem vypadat, je zobrazena na obrázku 1.1 nebo na stránkách [2] [3].



Obr. 1.1: Blokový diagram popisující jednoduchou pipeline pro vykreslení [20]

1.2 Scéna

Scéna je prostor definovaný souřadným systémem např. x, y ve dvourozměrné scéně (2D), který je zobrazen uživateli. Pokud chceme se scénou, nebo s jednotlivými objekty ve scéně, pohybovat nebo je přesunout, zvětšit, otočit, použijeme pro to vynásobení scény, objektu nebo vrcholu (vertexu) transformačními maticemi.

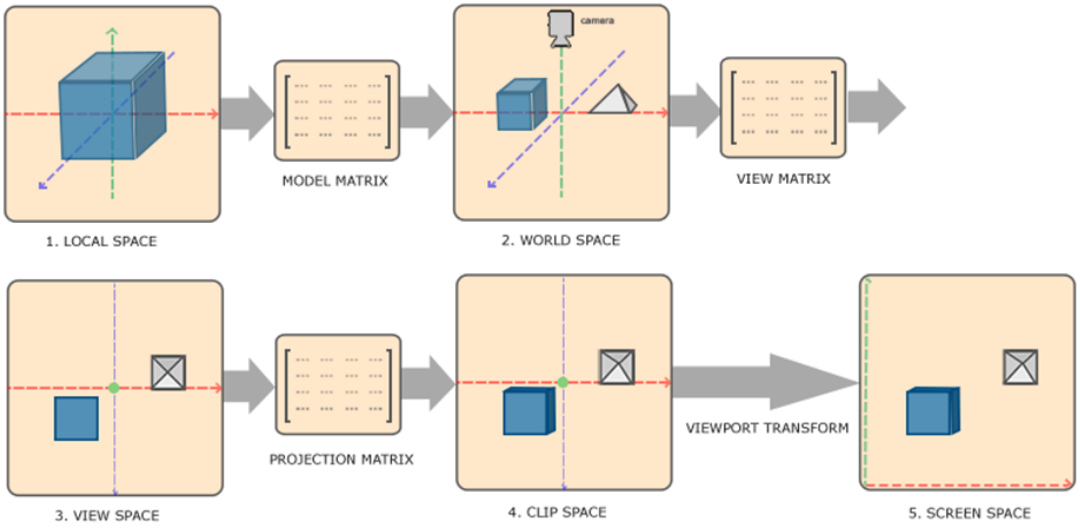
Nastavení kamery (projekční matice) je způsob, jak zobrazit trojrozměrný (3D) objekt na 2D výstup. Z pozice pozorovatele vychází projkční paprsky, které když protne objekt ve scéně, zobrazí ho na průmětnu (near clip space) [12].

Jak probíhá výpočet promítnutí objektu na scénu, je znázorněno na obrázku 1.2.

Popis jednotlivých souřadnic, na které se kamera scény právě dívá.

1. Místní souřadnice jsou souřadnice objektu ve vztahu k místnímu původu. Jsou to souřadnice, kde objekt začíná.
2. Dalším krokem je přeměnit lokální souřadnice na souřadnice světového prostoru, které jsou souřadnicemi pro větší svět. Tyto souřadnice jsou vztaženy k globálnímu původu světa spolu s mnoha dalšími objekty, které se rovněž vztahují k původu světa.

[2] https://www.root.cz/clanky/graficke-karty-a-graficke-akceleratory-21/
[3] https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview



Obr. 1.2: Blokový diagram postupu výpočtu promítaní objektů na scénu [12]

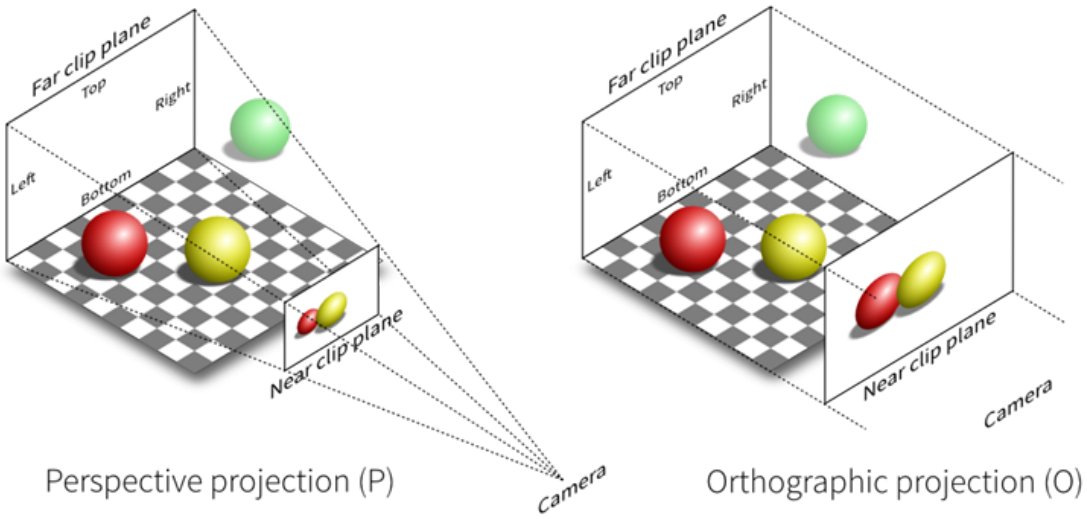
3. Dále transformujeme World space na souřadnice, odkud se na scénu díváme takovým způsobem, že každá souřadnice je viděna z pohledu kamery nebo diváka.
4. Nakonec přepočítáme místní souřadnice na souřadnice obrazovky, které nazýváme View port transformation. Ty přepočítají obraz do rozsahu -1.0 až 1.0. Výsledné souřadnice jsou pak odeslány do rasterizátoru, aby se změnily na fragmenty.

Tyto transformace jsou prováděny nad každým vrcholem ve scéně, proto je výhodné si tento výpočet provést jen jednou a výslednou maticí násobit jednotlivé vrcholy, ušetří se tím hodně výpočetního času.

Existuje mnoho různých způsobů, jak promítat 3D svazek na 2D obrazovku, nejčastěji se používá perspektivní a paralelní projekce, proto tyto dva druhy promítání budou dále rozebrány. Pro nastavení kamery a transformaci do souřadného systému kamery je třeba nastavit projekční matici(View Matrix).

1.2.1 Paralelní projekce

Paralelní projekce, někdy taky nazývaná ortografická, je projekce, kdy vzdálené objekty mají stejnou velikost jako blízké. Toho se využívá zejména v programech, kde nechceme žádné zkreslení reality, například technické aplikace jako (CAD, CAM, CAE) výkresové dokumentace, zobrazování 3D technických schémat atd. 1.3



Obr. 1.3: Názorné zobrazení scény, perspektivní a paralelní projekcí [12]

1.2.2 Perspektivní projekce

Paralelní projekce je ta kde, vzdálené objekty vypadají menší. Čehož se využívá pro virtuální realitu, architekturu, hry a další. Perspektivní projekce je pro člověka přirozenou projekcí, jelikož tímto způsobem vnímáme svět. 1.3

1.3 Základní transformace s objekty ve scéně

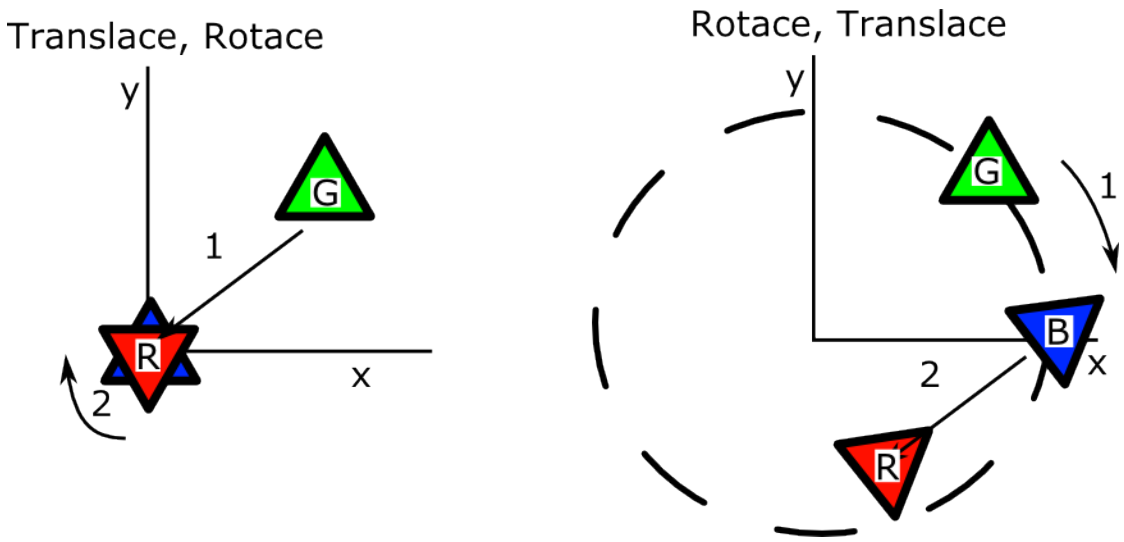
Existuje několik primitivních transformací, díky nimž dokážeme provádět základní přesuny, pro složitější přesuny je nutno využít více těchto matic. Je však velice důležité, jak matice naskládáme za sebe, jelikož násobení matic není komutativní. Pokud budeme nad skládáním přemýšlet, je důležité si uvědomit, že počátek souřadného systému je pro všechny objekty stejný a je tedy rozdíl, jestli objekt (například trojúhelník) prvně posuneme a pak otočíme nebo naopak. Pro znázornění je možné nahlédnout na obrázek 1.4. [14]

1.3.1 Posunutí

Posunutí ve 2D je proces přidání jiného vektoru k vektoru původnímu tak, aby vznikl nový vektor s odlišnou polohou než původní, čímž se počáteční bod (objekt) přesune na námi chtěné místo.

Vektor posunutí ve 2D

$$v = (xt, yt) \tag{1.1}$$



Obr. 1.4: Názorná ukázka prohození násobených matic

Výpočet nových souřadnic posunutého bodu.

$$x' = x + xt \tag{1.2a}$$

$$y' = y + yt \tag{1.2b}$$

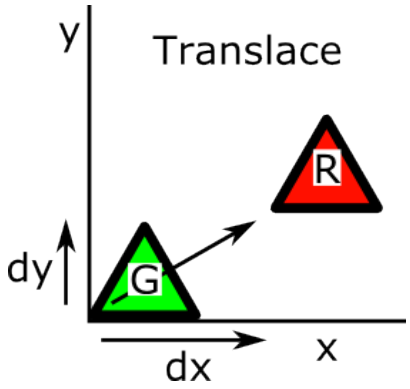
Maticoví zápis transformace v 2D.

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix} \tag{1.3}$$

Po roznásobení pozice základního objektu touto maticí se objekt, v tomto případě trojúhelník, posune o vzdálenost dx v ose x a o vzdálenost dy v ose y. Jak posunutí může ve 2D vypadat je znázorněno na obrázku. 1.5

Ve 3D je matice obdobná jako ve 2D, jen má o parametr dz navíc.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix} \tag{1.4}$$



Obr. 1.5: Znázornění translace

1.3.2 Rotace

Rotace ve 2D je změna úhlu otočení kolem počátku (středu souřadného systému), rotace posouvá jednotlivé objekty kolem středu po kruhu a to o vzdálenost úhlu α.

Výpočet nových souřadnic bodu rotace.

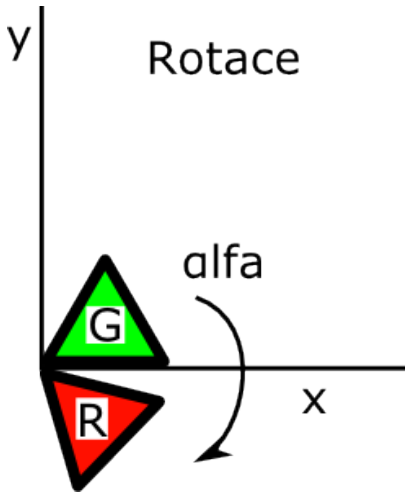
$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha \tag{1.5a}$$

$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha \tag{1.5b}$$

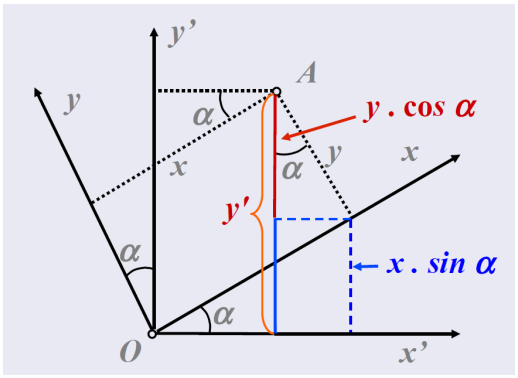
Maticoví zápis rotace ve 2D.

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

Po roznásobení pozice základního objektu touto maticí se objekt, v tomto případě trojúhelník, pootočí kolem středu souřadného systému o úhel alfa 1.6. Jak se posunutí provede je znázorněno na obrázku 1.7.



Obr. 1.6: Znázornění rotace



Obr. 1.7: Znázornění popisu výpočtu rotace

Ve 3D je situace trochu jiná. Pokud chceme tělesem rotovat, je třeba znát kolem, které osy bude tělo rotovat. Proto základní rotační matice ve 3D jsou 3 a to kolem osy x, y a z.

Rotace kolem osy X

$$Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

Rotace kolem osy Y

$$Ry = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.8}$$

Rotace kolem osy Z

$$Rz = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.9}$$

1.3.3 Změna měřítka

Změnu měřítka provádíme tehdy, pokud zachováme směr vektoru stejný a chceme pouze měnit jeho délku. Vzhledem k tomu, že pracujeme ve dvou nebo třech rozměrech, můžeme definovat měřítko pomocí dvou nebo trojrozměrného vektoru, přičemž můžeme měnit každou osu zvlášť a tak dosáhnou různě dlouhých os (x, y nebo z).

Výpočet nových souřadnic bodu změny měřítka.

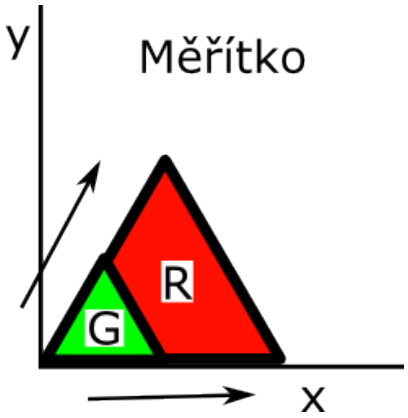
$$x' = x \cdot Sx \tag{1.10a}$$

$$y' = y \cdot Sy \tag{1.10b}$$

Maticový zápis změny měřítka ve 2D se změní, pokud Sx nebo Sy budou různé od jedničky. Pokud hodnota S je menší než 0, dochází k zrcadlení tělesa. Pokud S bude v intervalu od 0 do 1, daný objekt se zmenší. V posledním případě kdy S je větší než 1 se těleso zvětší stejně tak, jako je to znázorněno na obrázku 1.8.

Maticový zápis změny měřítka ve 2D.

$$S = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.11}$$



Obr. 1.8: Znázornění změny měřítka

Ve 3D je matice obdobná jako ve 2D, jen má víc o parametr Sz pro souřadnici z.

$$S = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

1.3.4 Zkosení

Zkosení, nebo někdy také označované shear, označuje míru zkosení tělesa ve směrech souřadného systému. Pokud se tedy jedná o 2D prostor, jsou to směry x a y. Pokud se jedná o trojrozměrnou scénu, je třeba vždy určit, v kterém směru má být daný objekt zkosen. Podle toho se vybere jedna ze tří matic zkosení 3D prostoru nebo jejich kombinace. Názorná ukázka zkosení je na obrázku 1.9.

Výpočet nových souřadnic bodu po zkosení objektu.

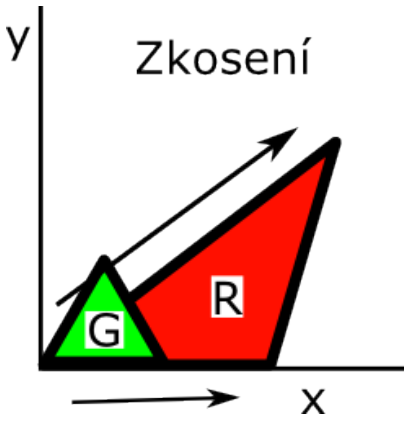
$$x' = x \cdot Shx \cdot y \tag{1.13a}$$

$$y' = y \cdot Shy \cdot x \tag{1.13b}$$

Maticoví zápis zkosení ve 2D.

$$Sh = \begin{bmatrix} 1 & Shx & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

Po roznásobení pozice základního objektu touto maticí se objekt, v tomto případě trojúhelník, zkosí o hodnoty Shx a Shy 1.9.



Obr. 1.9: Znázornění zkosení trojúhelníka

Ve 3D je situace trochu jiná než ve 2D. Jak už bylo naznačeno dříve, je třeba znát, ve kterém směru chceme objekt zkosit. Proto základní matice zkosení pro 3D jsou tři a to ve směru osy XY, YZ a XZ.

Matice zkosení popisující všechny směry.

$$S = \begin{bmatrix} 1 & Shxy & Shxy & 0 \\ Shxz & 1 & Shxz & 0 \\ Shxy & Shxy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.15}$$

2 X3D FORMAT (EXTENSIBLE 3D)

X3D je ISO standard XML formátu na ukládání 3D scén – geometrie a chování 3D objektů. Koncepce vychází z VRML97, důležitou změnou oproti VRML je možnost přejít z VRML syntaxe na XML syntaxi, avšak norma připouští obojí. Též definuje binární kódování X3D dat. [10]

Rozšíření X3D podporuje multi-stage (vícestupňové) a multi-texture rendndrování. Podporuje také stínování s lightmap a normalmap. Od roku 2010 X3D podporuje odložené vykreslování architektury. X3D může spolupracovat s dalšími open source standardy včetně XML, DOM a XPath. Standardizace X3D definuje několik profilů (sad dílů) pro různé úrovně schopností včetně X3D Core, X3D Interchange, X3D Interactive, X3D CADInterchange. Tvůrci prohlížeče mohou definovat vlastní rozšíření komponent před jejich předložením ke standardizaci ze strany Web3D Konsorciem. Formální revize a schvalování se pak provádí mezinárodní organizací pro normalizaci (ISO). Dohody o spolupráci jsou rovněž mezi Web3D konsorciem a Word Wide Web konsorciem(W3C), Open Geospecial konsorciem (OGC), Digital Imagine komunikací v medicíně (DICOM) a Khronos Group [10].

X3D formát je také podporován programy jako Matlab a Wolfram Alpha. Velkou výhodou je jeho použitelnost na internetu, jeho podporu zvládají všechny známé prohlížeče např. Internet Explorer, Google Chrome, Firefox, Safari a to jak pro PC, tak pro mobily. Pokud bychom se chtěli více zabývat aplikací X3D formátu pro HTML, bude nejlepší navštívit oficiální stránky X3Dom [1].

Jelikož X3D formát je tvořen na základě XML a VRML, lehce si tyto formáty představíme. Formát je také volně rozšiřitelný jazyk, což znamená, že si můžeme zavézt svoje tagy(značky), kterými si můžeme popsat například normě neznámí objekt [7].

2.1 VRML

VRML (Virtual Reality Modeling Language) je grafický formát založený na deklarativním programovacím jazyce (co se má udělat, a ne, jak se to má udělat), který byl navržen především pro popis trojrozměrných scén obsahujících aktivní i pasivní objekty použité například v aplikacích virtuální reality. Nejedná se o jediný formát (či jazyk) této kategorie, dnes se například poměrně razantním způsobem prosazuje formát X3D, který lze chápat jako ideového nástupce VRML a v minulosti si prakticky každá firma vytvářející 3D aplikace navrhla vlastní formát, ovšem doposud

[1] `https://www.x3dom.org/`

se z grafických formátů a deklarativních jazyků určených pro popis virtuální reality nejvíce rozšířil právě jazyk VRML [8].

Pro představu, jak může VRML kod vypadat, si můžeme prohlédnout přílohu B.2, která popisuje orientaci kamery a ve scéně popisuje krychli a dvě koule, jenž mají navíc zadaný materiál a posunutí.

2.2 XML

XML (Extensible Markup Language) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Je zjednodušenou podobou staršího jazyka SGML. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat. Používá se pro serializaci dat, v čemž soupeří např. s JSON nebo YAML. Zpracování XML je podporováno řadou nástrojů a programovacích jazyků.

Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých popisuje strukturu z hlediska věcného obsahu jednotlivých částí, ale nezabývá se vzhledem. Prezentace dokumentu (vzhled) může být definována pomocí kaskádových stylů. Další možností zpracování je transformace do jiného typu dokumentu nebo do jiné aplikace XML [9].

Jelikož Xml jazyk je volně rozšiřitelný, nemá ani přesně definované svoje tagy (značky), ty si musí každý uživatel zadefinovat sám. Ale abychom si tento jazyk nepředstavovali jako nějaký náhodně vygenerovaný text, tak i tento jazyk má svá pravidla, která si teď představíme. Tyto pravidla dodržuje i jazyk X3D [2].

2.2.1 Syntaktická pravidla Xml

Syntaktická pravidla XML jsou velice jednoduchá k naučení a používání, ale také velmi přísná. Taktéž vytvořit software k manipulaci s XML není programátorsky nikterak obtížné. Abychom si mohli vysvětlit jednotlivá pravidla, nejdříve si popíšeme jak jednoduchý Xml dokument může vypadat. [15] [16]

Výpis 2.1: Ukázka Xml formátu.

```
1 <?xml version="1.0" encoding="ISO-8859-2"?>
2 <mail>
3   <pro>Pepu</pro>
4   <od>Ondry</od>
5   <predmet>Připomínka</predmet>
6   <text>V kolik je ten sraz?</text>
```

[2] https://www.w3schools.com/xml/default.asp

```
7 </mail>
```

První řádek deklaruje, jaká verze Xml bude použita a jakou kódovou sadu budeme používat, v našem případě středoevropský jazyk(ISO-8859-2). Druhý řádek udává rodičovský element, který je vždy jen jeden. V našem případě říká, že se bude jednat o mail. Další čtyři řádky jsou potomci mailu, jsou to elementy podřízené rodiči. Poslední řádek mailu ukončuje konec rodičovského elementu.

Pravidla:

- Každé Xml musí mít uzavírací tag, v našem případě jsou to tagy:

```
1 </pro>, </od>, </mail>
```

- Xml rozlišuje malá a velká písmena.

```
1 <Mail> chybný zápis </mail>
2 <mail> správný zápis </mail>
```

- Záleží na pořadí, v kterém napíšeme jednotlivá jména elementů.

```
1 <mail> <Pepovi> chybný zápis  </mail> </Pepovi>
2 <mail> <Pepovi> správný zápis </Pepovi> </mail>
```

- Xml musí mít hlavní element rodiče.

```
1 <rodic>
2   <potomek>
3     <podpotomek>...</podpotomek>
4   </potomek>
5 </rodic>
```

- Hodnoty Atributu musí být uchovány v 'jednoduchých' nebo v "dvojitých"uvozovkách. Atribut nese doplňující informace k elementu.

```
1 <mail DATE = 5.11.2010> chybný zápis </mail>
2 <mail DATE = "5.11.2010"> správný zápis </mail>
3 <mail DATE = '5.11.2010'> správný zápis </mail>
```

- Xml redukuje mezery mezi znaky, je tedy jedno, jestli do věty napíšeme více mezer mezi slova, jelikož Xml formát nám je zredukuje, což může být výhodou i nevýhodou.
- Xml komentáře se píší stejně jako komentáře jazyka HTML.
  <!—komentář – >
- Všechny elementy mají mezi sebou vztah. Element je vše od začátku do konce tagu včetně. Každý element může obsahovat: smíšené hodnoty, jednoduché hodnoty(text), prázdnou hodnotu, atribut.

Výpis 2.2: Příklad X3D formátu.

```
<X3D  id="boxes" showStat="false" showLog="false"
x="0px" y="0px" width="500px" height="500px">
    <Scene>
        <Viewpoint position="10 20 30"
        orientation="0 0 0">
        </Viewpoint>
        <Shape DEF="boxShape">
        <Box DEF="box"></Box>
        </Shape>
    </Scene>
</X3D>
```

- Jméno elementu nesmí začínat číslem nebo interpunkčním znakem. Nesmí obsahovat mezery a nesmí začínat znaky xml, Xml a XML. Název může vypadat jakkoli dlouhý či krátký, ale je nutné, aby dodržoval již dříve zmiňovaná pravidla. Je i velice vhodné nepsat v názvu interpunkční znaky.

2.3 X3D syntaxe

Syntaxe jazyka X3D je stejná jako u jazyka Xml a také pro ni platí stejná pravidla. Tento jazyk, jak už dříve bylo řečeno, slouží pro uchování 3D scén, geometrie a chování objektu v 3D, čehož bylo v mém projektu využito. Proto bude zápis geometrického kloubu a dalších elementů popsán v dalších kapitolách. Základní zápis scény v X3D formátu.

Na začátku je stejně jako definice verze Xml formátu a popis, jaká znaková sada bude pro kódování následujícího dokumentu použita. Na druhém řádku je typ dokumentu a adresa, kde byl zdrojový kód vytvořen, a je tam také napsáno s jakou verzí X3D formátu se pracuje. Značka <X3D> označuje mateřský element pro 3D scénu(root element), v jeho atributech si můžeme všimnout, že dokáže popsat šířku, výšku plátna, identifikační označení a mnoho dalších informací pro inicializaci vykreslovaného okna. Pro tuto práci je však nejdůležitějším elementem celého X3D formátu element <Scene>, který popisuje chování celé scény a všech jeho objektů kloubu či dalších vlastností, které scéna může mít. V tomto příkladě scény můžeme vidět, že obsahuje tři potomky a to <Viewpoint>, <Shape>, <Shape>. Potomek <Viewpoint> má dva velice důležité atributy position a orientation, přičemž v těchto atributech najdeme jednoduché pozice, odkud a kam má kamera scény směřovat.

V potomku <Shape> bude nalezen object (primitivum), který definuje, co je daný object zač. V tomto případě se jedná o krabici (box), která nemá definované žádné bližší parametry. Lze tedy říct, že se bude jednat nejspíše o box s jednotkovými rozměry.

2.3.1 Elementy popisující scénu

Pokud bych chtěl podrobněji popsat, jak daný model <shape> bude vypadat, je třeba mu zadefinovat mnoho dalších vlastností kromě rozměrů a to, jaký na něj bude použit materiál, kde ve scéně bude daný objekt umístěn a mnoho dalších elementů popisujících vlastnosti objektu a scény. Abych však nemusel vysvětlovat vše, co je možné v X3D definovat, zaměřím se pouze na popis základních elementů pro vytvoření spojeného objektu a to <Shape> a <Joint> . Další věc, na kterou je důležité upozornit je fakt, že v této práci bylo pracováno jen s tuhými tělesy, proto nad každým shapem je element <RigidBody>, který určuje, že se jedná o tuhé těleso. [19]

Shape Nejdůležitějším popisem v jakémkoli 3D jazyce je popis tělesa, které má být ve scéně zobrazeno, ať už se jedná o nějaký složitý popis tělesa (například lebky) nebo se jedná o základní element jako je třeba krabice (box). Pokud by byl objekt složitější, byl by definován pomocí pozic vrcholů daného objektu. Tato práce se však bude zabývat jen takzvanými primitivy, což jsou objekty (<Box>, koule <Sphere>, válec <Cylinder>, kužel <Cone>), které když složíme dohromady, dokážeme popsat jakékoli složité těleso.
Jak vypadá popis tělesa v X3D formátu, je popsáno a ukázáno v příkladu. V tomto příkladu je popsán <Shape> krabice <Box>, jenž má také zadefinovanou barvu celého tělesa. Mateřským elementem shapu je <CollidableShape>, jenž vysvětuluje fakt, že těleso bude zařazeno do těles, jenž mohou mít mezi sebou kolizi.

Výpis 2.3: Příklad popisu RigidBody v X3D formátu.

```
1 <CollidableShape DEF='BOX'>
2   <Shape containerField='shape'>
3     <Appearance>
4       <Material emissiveColor='0.0 0.0 1.0'/>
5     </Appearance>
6     <Box size='0.3 0.02 0.02'/>
7   </Shape>
8 </CollidableShape>
```

Výpis 2.4: Příklad popisu RigidBody v X3D formátu.

```
<RigidBody DEF='element15' RotationX='PI / 4'
Translation='1 0 0' mass='0'>
  <CollidableShape USE='elementShape2'/>
</RigidBody>
```

Jak už bylo řešeno v tomto případě budou definovány pouze tuhá tělesa, proto popíšu, jak takové tuhé těleso v X3D může být popsáno. Slouží k tomu klíčové slovo (element) <RigidBody>, ve kterém jsou definovány atributy DEF, RotationX, Translation a mass. Atribut DEF říká, jak se vytvořené tuhé těleso bude jmenovat. RotationX říká, o jaký úhel kolem osy X se těleso otočí, práce s rotačními maticemi byly popsány již dříve 1.3.2. Translation udává, kde bude dané těleso vytvořeno a o kolik se posune od středu souřadného systému1.3.1. A poslední definice mass udává, jak moc je těleso hmotné a jak moc na něj bude působit gravitační síla. V tomto případě je mass=0, bude se tedy jednat o statické těleso.

Joint Pokud se bavíme o 3D grafice, jedním z dalších věcí, které se na pohybující objekt uplatňují, je spojení dvou uzlů kloubem <Joint>. Toto spojení může mít několik podob a to kloub otáčející se do všech směrů souřadného systému <BallJoint>, otočný kloub <HingeJoint> nebo posuvný kloub <SliderJoint>. Jak popis prvních dvou kloubu vypadá, je možné vidět na ukázkách kódu 2.3.1 2.3.1 a popis jejich funkčnosti, je popsáno v kapitolách 3.2.2, 3.2.3 a 3.2.1.

Výpis 2.5: Příklad popisu BallJointu v X3D.

```
<BallJoint anchorPoint='0.15 0.0 0.0'
containerField='joints' forceOutput='"NONE"'>
  <RigidBody USE='BODY-1' containerField='body1'/>
  <RigidBody USE='BODY-2' containerField='body2'/>
</BallJoint>
```

Výpis 2.6: Příklad popisu HingeJointu v X3D.

```
<SingleAxisHingeJoint anchorPoint='0.15 0.0 0.0'
containerField='joints' forceOutput='"NONE"'>
  <RigidBody USE='BODY-1' containerField='body1'/>
  <RigidBody USE='BODY-2' containerField='body2'/>
</SingleAxisHingeJoint>
```

3 BULLET PHYSICS

Bullet physics je profesionální open source knihovna psaná v jazyce C++, která řeší detekci kolize tuhých a pružných objektů. Tato knihovna je primárně tvořená pro využití v hrách, vizuální efekty a robotické simulace. Bullet physics umožnuje:

- Práci s tuhými (Rigid body), pružnými (soft body) tělesy a detekcí diskrétní a kontinuální kolizí.
- Kolidující objekty (Collision shapes) zahrnují kouli, krabici, válec, kužel a konvexní obálku libovolného objektu.
- Pružná tělesa (Soft body) nám umožňují vytvořit tkaninu, lano a libovolné deformovatelné předměty.
- Nastavení tuhým a pružným tělesům omezení pohybu, propojení objektů díky uzlům a vytvoření motoru, který vyvíjí sílu působící na dané těleso v námi zvoleném směru.

3.1 Vytvoření primitiv

V bullet physic můžeme vytvořit dva druhy těles: měkká tělesa (Softbody) a tuhá tělesa (Rigidbodie). V tomto projektu se budu zabývat pouze tuhými objekty. Pokud bychom chtěli vědět, jak fungují měkká tělesa, můžeme se podívat do manuálu [17].

Pro vytvoření takového objektu slouží zápis.

```
1 LocalCreateRigidBody(float mass, Matrix
2 startTransform, CollisionShape shape);
```

Údaj mass udává, jak bude daný objekt hmotný. Pokud toto číslo bude nula, bude se jednat o statický objekt, na který nepůsobí gravitace. StartTransform říká, jak bude objekt transformován oproti středu souřadného systému. Na objektu je možné uplatnit jednoduché transformace, viz. 1.3 shape. Objekt říká, jaké primitivum bude vykresleno, bullet physic umí vykreslit několik základních druhů objektů. Zde jsou některé z nich.

BoxShape Krabice může být definována, buďto pomocí jednoho parametru, v tom případě se bude jednat o krychli, nebo pomocí tří parametru, pak se bude jednat o kvádr.

```
1 %CollisionShape shape = new BoxShape(float a,
2 float b,float c);
```

SphereShape Koule je definována pouze jedním číslem a to je její poloměr od středu.

```
1 %CollisionShape a = new SphereShape(float radius)
```

CylinderShape Válec je definován třemi hodnotami, dvě hodnoty udávají rádius a prostřední hodnota udává výšku válce.

```
%CollisionShape a = new CylinderShape(float
halfExtentsX,float halfExtentsY,
float halfExtentsZ);
```

ConeShape Kužel je definován dvěma hodnotami, poloměrem podstavy a výškou.

```
%CollisionShape a = new ConeShape(float radius,
float height);
```

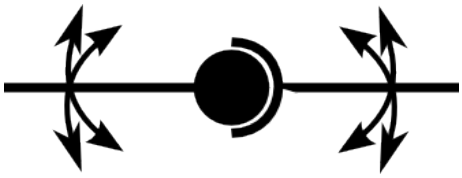
3.2 Joint

Stejně jako v X3D formátu 2.3.1 i tady jsou popsány základní klouby a to ball, hinge a slider. Jak si tyto spoje můžeme představit a jak se popisují v C# bude vysvětleno v následujících kapitolách. Důležité je vědět, že i když klouby mají určitý stupeň volnosti, lze jej omezit v nějakém rozsahu, příkladem může být omezení slideru, aby objekt neutekl mimo chtěný rozsah.

3.2.1 Obecný kloub

Obecný kloub, neboli Point to Point constraint, nám udává pomocí dvou bodů, kde budou tělesa spojeny kloubem. Tento druh se chová jako klasický lidský kloub. Tento kloub má tři stupně volnosti a to rotaci kolem osy x, y a z, díky tomu můžeme propojit řetězec více objektů. Jak si daný kloub představit, je znázorněno na obrázku 3.1. Jak se kloub definuje v C#, je znázorněno v kódu níže.

```
%Point2PointConstraint(btRigidBody& rbA,const
btVector3& pivotInA);
%Point2PointConstraint(btRigidBody& rbA,
btRigidBody& rbB, const btVector3& pivotInA,
const btVector3& pivotInB);
```

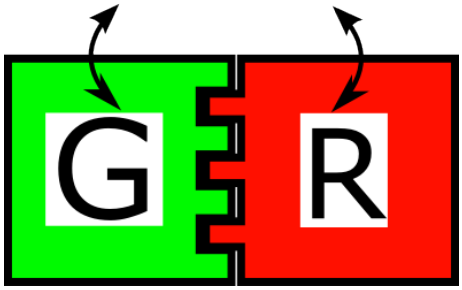


Obr. 3.1: Znázornění obecného kloubu

3.2.2 Závěsný kloub

Závěsný kloub, neboli Hinge Constraint, omezuje další dva stupně volnosti. Tímto kloubem můžeme tedy otáčet jen kolem jedné osy, jak je tomu znázorněno na obrázku. Pokud bude červený objekt statický, zelenému objektu bude umožněn jen kruhový pohyb kolem kloubu. Tohoto závěsu se využívá pro upevnění dveří nebo kol. Jak si daný kloub představit je znázorněno na obrázku 3.2. Druhy definic kloubu v C# jsou znázorněny v kódu níže.

```
1 HingeConstraint(btRigidBody& rbA,btRigidBody& rbB,
2 const btTransform& rbAFrame, const btTransform&
3 rbBFrame, bool, useReferenceFrameA = false);
```



Obr. 3.2: Znázornění závěsného kloubu

3.2.3 Posuvný spoj

Posuvný spoj, neboli Slider Constraint, umožňuje tělu otáčet se kolem jedné osy a posouvat se podél jedné osy. Jak si daný kloub představit, je znázorněno na obrázku 3.3. Jak se kloub definuje v C#, je znázorněno v kódu níže.

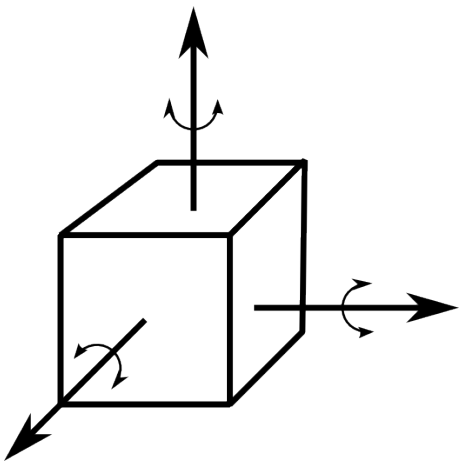
```
1 SliderConstraint(btRigidBody& rbA, btRigidBody& rbB,
2 const btTransform& frameInA, const btTransform&
3 frameInB ,bool, useLinearReferenceFrameA);
```

Obr. 3.3: Znázornění posuvného spoje

3.3 Omezení pohybu v šesti stupních volnosti

Omezení pohybu v šesti stupních volnosti(6Dof Constraint) - toto obecné omezení může napodobit řadu standardních omezení, a to konfigurováním každého z šesti stupňů volnosti. První tři osy jsou lineární, ty představují translaci tuhých těles, zbylé tři udávají rotaci kolem těchto os, představují rotační pohyb. Každá osa může být buď uzamčena, volná nebo omezená. Při konstrukci nového Generic6DofSpring2Constraint jsou všechny osy zamčené. Jak si takové těleso představit, je znázorněno na obrázku 3.4. Jak se šest stupňů volnosti definuje v C#, je znázorněno v kódu níže.

```
%btGeneric6DofConstraint(btRigidBody& rbA,
btRigidBody& rbB, const btTransform& frameInA,
const btTransform& frameInB ,bool
useLinearReferenceFrameA);
%Vector3 lowerSliderLimit = btVector3(-10,0,0);
%Vector3 hiSliderLimit = btVector3(10,0,0);
```

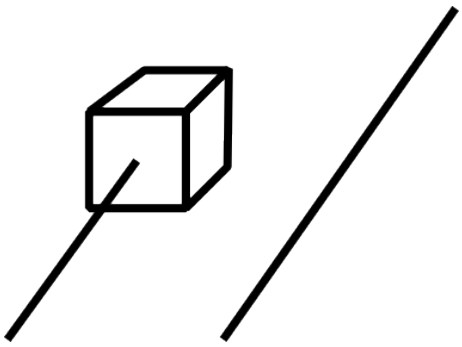


Obr. 3.4: Znázornění šesti stupňů volnosti

3.4 Raytracer

Raytracer je nástroj pro realizaci snímače světelného paprsku(laseru). Ve virtuální scéně se jedná o jakési provedení optického snímače. Tento paprsek reaguje na kolizi s jakýmkoli hmotným tělesem. To zahrnuje okraj tělesa i jeho trup. Jak Raytracer definuje v C#, je znázorněno v kódu níže. Znázornění kolize paprsku s objektem je na obrázku 3.5.

```
%ClosestRayResultCallback a = new
 ClosestRayResultCallback(ref Vector3 rayFromWorld,
ref Vector3 rayToWorld);
```



Obr. 3.5: Znázornění vyhodnocení kolize ray testu

3.5 Motor

Bullet physice umožňuje také na každý hmotný objekt, který má nastavené omezení pohybu. Ať už je to kloub nebo nejčastěji používaný 6Dof Constraint, ve kterém je možné omezit pohyb podle potřeby, viz kapitola 3.3. Pro nastavení motoru slouží tři hodnoty: spuštění motoru (EngineSet), tato hodnota udává, zda-li je motor zapnutý nebo vypnutý. Nastavení rychlosti (EngineVelocity), jakou chceme, aby se motor otáčel nebo posouval, a poslední je síla motoru (MotorForce). Jelikož EngineVelocity a MotorForce se zadávají vektorem, je třeba, aby oba dva vektory měli stejný směr a lišili se pouze velikostí. Pokud toto pravidlo nebude dodrženo, knihovna bullet physic nebude fungovat korektně.

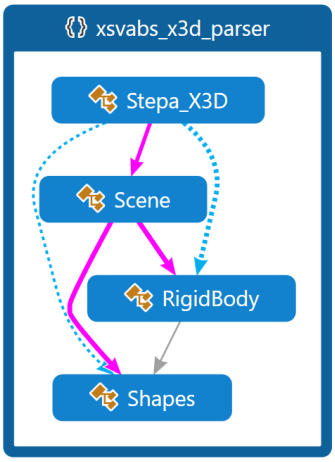
Příklad užití motoru na 6Dof Constraint:

```
public override void OnUpdate(){
Generic6Dof.TranslationalLimitMotor.EnableMotor[0]
= EngineSet;
Generic6Dof.TranslationalLimitMotor.TargetVelocity
= EngineVelocity;
Generic6Dof.TranslationalLimitMotor.MaxMotorForce
= MotorForce;}
```

Metody uplatňující sílu působící na objekt(motor) se nejčastěji používají ve třídách, které se stále přepočítávají. V tomto příkladě je to metoda OnUpdate().

4 X3D PARSER

X3D Parser je metoda, která umožní otevřít X3D formát, vybrat z něj data (vyparsovat) a následně je vytvoří v Bullet Phizice. V tomto případě pro parsování byla vytvořena speciální třída Scene, na kterou když se zavolá metoda Parse, tak nám vybere data ze zvoleného X3D dokumentu do vytvořené struktury 4.2. Tato struktura funguje tak, že nejdříve je ve třídě Stapa_X3D vytvořena třída Scene, na kterou se následně zavolá metoda parse, čímž se spustí rekurzivní parsovací metoda, která vyhledává data,jenž chceme z dokumentu vyparsovat. Pokud tam nejsou, a jsou důležitá pro scénu jako například eye_position, jsou tyto hodnoty přiřazeny implicitně. Jak jsou jednotlivé třídy provázány, je znázorněno na obrázku 4.1.



Obr. 4.1: Mapa kódu tříd parseru

4.0.1 Třída Scene

Tato třída nám zajišťuje dvě nejdůležitější části: metodu Spoj a parsovací metodu, která načte X3D dokument. V tomto případě je to dokument belt.xml, ukázka tohoto kódu je v příloze B.1. Když má dokument koncovku xml, je tomu z důvodu, jak už bylo řečeno dříve, že X3D formát vychází ze syntaxe, proto dokument může mít koncovku xml, i když je v něm popis X3D scény.

Postup mé parsovací metody.

1. Vytvoří se XmlDocument.

```
1 XmlDocument doc = new XmlDocument();
```

2. Do dokumentu se nahrají z externího dokumentu belt.xml, který se nachází přímo u spustitelného souboru (xsvabs_x3d_parser.exe). Ukázka toho, jak parsovaný dokument vypadá, je v příloze B.1.

```
1 doc.Load("belt.xml");
```

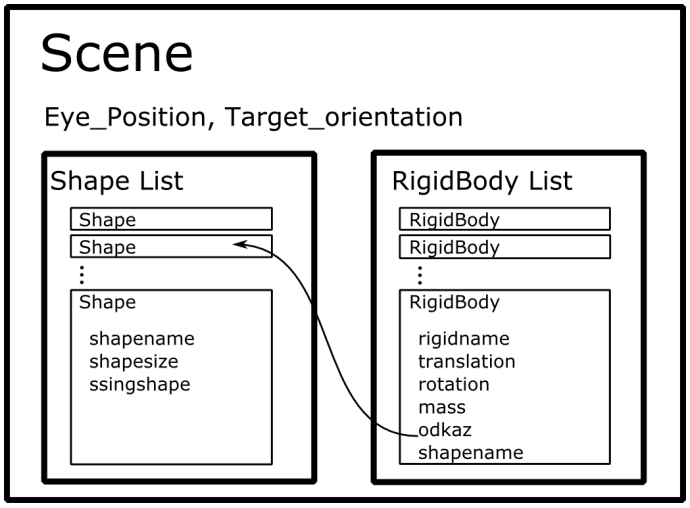
3. Zavolá se funkce parse_node s nahraným dokumentem na vstupu, ve které se dokument rekurzivně prohledává.

```
1 parse_node(doc.DocumentElement);
```

4. Pokud rekurzivní prohledávání dokumentu najde element Viewpoint, vybere z něj hodnoty atributu position, orientation a zapíše je do svých proměnných eye_position, target_orientation.
5. Pokud jméno elementu bude Shape, vytvoří se nová, která si sama vybere data, která potřebuje 4.0.2. V tomto případě je to Shape se jménem elementShape 1-8 a Shape belt viz. B.1.
6. Pokud jméno elementu bude RigidBody, vytvoří se nové RigidBody, které si samo vybere data, která potřebuje 4.0.3.
7. Posledním krokem se zavolá funkce Spoj, která spojí daný Shapes s RigidBody. Tato funkce projde všechny RigidBody a přiřadí mu příslušný Shapes podle jména shapu, které RigidBody vyparsuje. V tomto případě se vyparsují RigidBody se jménem element 1-20 viz B.1.

```
1 if (actrigid.shapename == actshape.shapename)
2     actrigid.odkaz = actshape;
```

Jak vypadají vyparsovaná data, je znázorněno na obrázku 4.2.



Obr. 4.2: Znázornění struktury vyparsovaných dat

4.0.2 Třída Shapes

Třída Shape slouží pro vytvoření proměnných a vyparsování hodnot Shapu a to stringu shapename, Vector3 shapesize a int UsingShape. Velikost tvořeného Shapu shapesize nabývá tří hodnot, proto díky tomuto vektoru dokážeme definovat rozměry jakéhokoli primitivního tělesa(žádné těleso nepotřebuje pro vytvoření více než tři hodnoty) a poslední hodnoty, které je nutno vyparsovat, je informace, o jaký se jedná shape. Zda-li je to box, sphere, cone nebo cylinder, podle toho bude do hodnoty shapeusing přiřazena hodnota 1,2,3,4. Pokud by se jednalo o neznámé těleso, byla by tato hodnota nastavena na nulu.

Postup vyparsování jednotlivých hodnot třídy Shape.

1. Z předchozí třídy Scene byla zavolána metoda ShapeParse, která slouží k vyparsování listu jednotlivých hodnot Shapu, jak je znázorněno na obrázku 4.2.
2. Metoda ShapeParse hledá element CollidableShape. Pokud ho najde, vyčte si jeho atribut DEF a uloží si jeho hodnotu, která říká, jak se vytvořený Shape bude jmenovat.
3. V dalším kroku se zjišťuje, zda-li se jedná o box, sphere, cone nebo cylinder. Podle toho se nastaví hodnota shapeusing a vyčtou se rozměry daného tělesa shapesize.
4. V předchozím kroku byly vyčteny všechny potřebné informace, takže parser může pokračovat parsováním dalších objektů.

4.0.3 Třída RigidBody

Třída RigidBody slouží pro vytvoření proměnných a vyparsování hodnot RigidBody, a to rigidname, transformation, rotation, mass, odkaz a shapename. Proměnná rigidname udává, jak se dané RigidBody bude jmenovat. Transformation je trojrozměrný vektor Vector3, díky kterému bude těleso přesunuto na zadanou pozici. Pokud se tato hodnota v X3D formátu nenachází, jsou vektoru transformation přiřazeny hodnoty (0,0,0), díky kterým víme, že se daný objekt vytvoří ve středu souřadného systému. Další potřebnou hodnotou pro vytvoření objektu je rotation, ta udává, o kolik radiánů se daný objekt otočí kolem jednotlivých os. rotation je v kódu popsán jako rotationX, rotationY, rotationZ a každá hodnota udává rotaci kolem své osy viz. 1.3.2. Hodnota mass říká, jak moc bude tuhý objekt hmotný. Je-li číslo větší než nula, na objekt působí gravitační síla a posouvá se v tomto případě ve směru osy y pokud hodnota čísla mass bude rovna 0,jedná se o nehmotné a tedy statické těleso. Poslední a jednou z nejdůležitějších hodnot, které se vyparsují, je shapename, díky kterému jsem schopni aktuálnímu RigidBody následně přiřadit příslušný Shape.

Postup vyparsování jednotlivých hodnot třídy RigidBody.

1. Z předchozí třídy Scene byla zavolána metoda RigidBody, která slouží k vyparsování listu jednotlivých hodnot RigidBody, jak je znázorněno na obrázku 4.2.
2. V prvním kroku se hledá atribut RotationX. Jeho hodnota stringu je předána do metody rotationfunction(), která zjistí, jaká hodnota se uvnitř nachází a podle toho vrátí příslušnou hodnotu. Pokud jméno není definovano je vrácena hodnota 2, která udává, že na dané těleso nebude použita rotace kolem osy x. Tento postup je uplatněn i na atribut RotationY a RotationZ.
3. Dále se vyparsuje hodnota atributu Translation a nahraje se do proměnné rigidtranslation. Pokud se tento atribut v aktuálním RigidBody nenachází, jsou proměnné rigidtranslation přiřazeny hodnoty (0,0,0), jak už bylo řečeno dříve.
4. Dalšími kroky se pak vyberou hodnoty mass a shapename. Co tyto hodnoty říkají, je popsáno v odstavci 4.0.3.
5. V předchozím kroku byly vyčteny všechny potřebné informace o RigidBody, takže metoda RigidParse() se ukončí.

4.1 Vytvoření těles do scény

Jakmile máme již vyparovaná data, je třeba objekty vykreslit ve scéně. K tomu nám slouží funkce, která zajisťuje, že každé tuhé těleso(RigidBody), které bylo vytvořeno, bude vykresleno. K těmto účelům byl vybrán cyklus foreach, jenž každým RigidBody zapíše do scény na jeho správné místo. Jak tato funkce vypadá, je znázorněno níže.

```
1  if (actrigid.shapename == actshape.shapename)
2      actrigid.odkaz = actshape;
3  foreach(RigidBody a in X3D.rigidbody)
4    {
5    if (a.odkaz != null)
6    {
7      var elementShape = new BoxShape(a.odkaz.shapesize);
8      var element = LocalCreateRigidBody(0,
9      Matrix.Translation(a.rigidtranslation)
10     * Matrix.RotationX((float)+Math.PI * a.rotationX) *
11     Matrix.RotationY((float)+Math.PI * a.rotationY) *
12     Matrix.RotationZ((float)+Math.PI * a.rotationZ),
13     elementShape1);
14     }
15   else;
```

4.2 Zhodnocení

V této části projektu byl zrealizován parser X3D formátu, který vybere data z dokumentu B.1 a vytvoří z něj scénu viz. příloha C.1.

Když se vrátím ještě do části práce 4.1, můžete si zde povšimnout, že se před vytvořením tělesa ptám, jestli není odkaz na shape null. Je to proto, že při propojování těles se občas z neznámých důvodů tělesa nepropojí a nemají odkaz na svůj shape. Je však s podivem, že i když program hlásí, že RigidBody nemá odkaz na shape, stejně se vytvoří všechny nadefinované objekty. I když tyto chyby nemají vliv na funkčnost, bylo by dobré je odstranit.

Pokud bychom chtěli tento parser rozšířit, bylo by dobré vytvořit několik dalších listů, které by vybraly ze scény například klouby, Ray testy nebo popis motoru. Co se týká kloubu, jsou v této práci uvedeny příklady, jak se definují. U popisu Ray testu nebo motoru nastává problém, jak je definovat, jelikož na webu [1] nějaké ukázky popisu motoru jsou, ale moc se neshodují s popisem motoru v bullet phyzic. Proto by bylo dobré buď najít jiný vhodný popis a nebo si vytvořit svůj vlastní. S Ray testem je to obdobné jako s motorem. Pokud by byly dodefinovány a vyparsovány všechny tyto elementy, bylo by možno popsat většinu simulátorů pracujících s RigidBody.

Další možností rozšíření projektu by bylo, pustit se do tvorby měkkých těles (SoftBody). Tímto tématem se ovšem tato práce nezabývala, tudíž jak se tato tělesa vytváří, by bylo nutno nahlédnout do jiných prací.

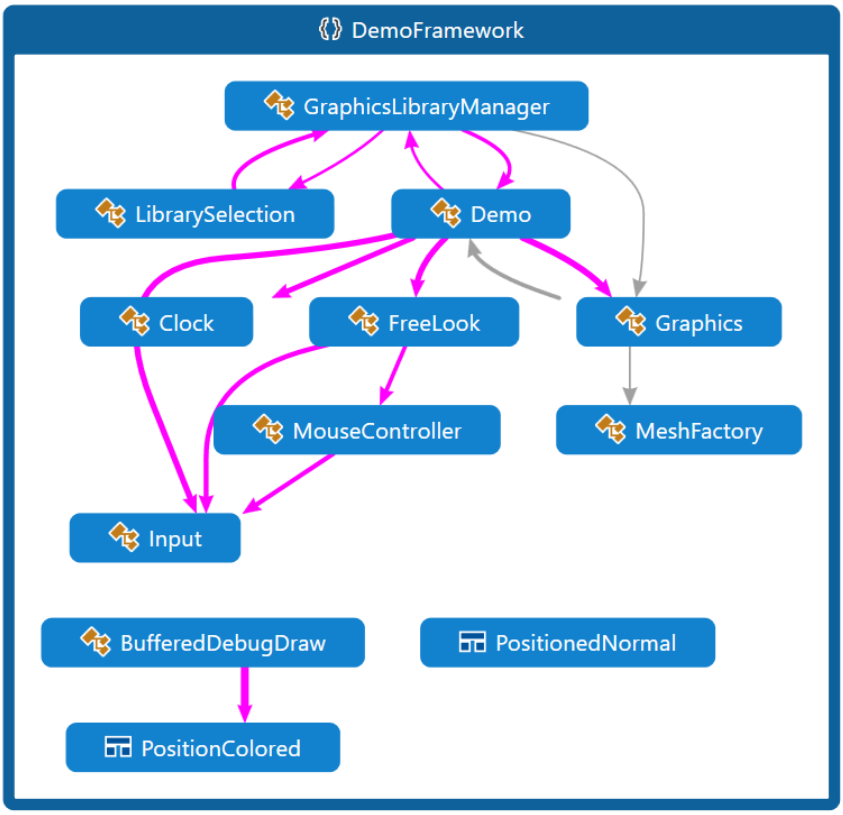
[1] `http://www.web3d.org/x3d/content/examples/Basic/RigidBodyPhysics/`

5 SIMULÁTOR LABORATORNÍHO MODELU

V mém projektu byla využita verze Bullet physic verze 2.83 [17], která je naimplementována v balíčku, jenž je volně ke stažení na stránkách [1]. Pro užívání tohoto balíčku je využíván .NET framework, jenž nám umožní jednotlivé komponenty bullet physic vykreslit do scény. Pro realizaci projektu byl tedy využit už existující framework viz. kapitola5.1.

5.1 .NET Framework

Pro vykreslení scény do Windows formuláře a nastavení jeho parametru, byla využita třída Demo, jenž vytvoří .NET Framework aplikaci a nastaví rozměry scény, do které se bude rendrovat daná simulace. Pro rendrování scény potom Demoframework může využít knihoven DirectX 11, DirectX 10, DirectX 9 nebo OpenGL. Jak vypadá mapa kódu jmenného prostoru DemoFramework, je vidět na obrázku 5.1. [18]



Obr. 5.1: Mapa kódu jmenného prostoru DemoFramework

[1] https://andrestraks.github.io/BulletSharp/

Pro aplikace jsou v tomto jmenném prostoru nejdůležitější třídy Demo, Graphic a FreeLook, kde třída Demo slouží jako propojení jednotlivých částí windows aplikace a umožňuje do scény přidávat nové objekty a animace. Dále také zajišťuje práci s jednotlivými událostmi, ať už jde o vnitřní záležitost, jako například kolize dvou objektů nebo o externí záležitost, jako je třeba zmáčknutí jakékoli klávesy. Třída Graphic slouží pro vytvoření okna Windows formuláře, předchystává projekční matice pro renderer a stará se o výpis textu do formuláře (například vypisuje, kolika rámci za sekundu). Scéna vykresluje a nastavuje, jak přesně bude dané okno Windows formuláře aplikace vypadat. Třída FreeLook zajišťuje volný pohyb kamery po scéně, což je zajištěno díky držení levého tlačítka myši a pohybu s ní a kláves W, S, A, D. Jak jsou všechny třídy mezi sebou propojeny, je vidět na obrázku 5.1.

5.1.1 Nový projekt

Díky najití již existujících volně šiřitelných examplů, které zajišťují vytvoření Windows formulář aplikace a rendrování objektu do scény. Této práce jsem byl ušetřen a mohl jsem se více věnovat pochopení, jak se v bullet physic definují jednotlivé pohyby a objekty, což bude popsáno v dalších kapitolách. [2]

Postup, jak vytvořit nový projekt využívající třídy Demo.

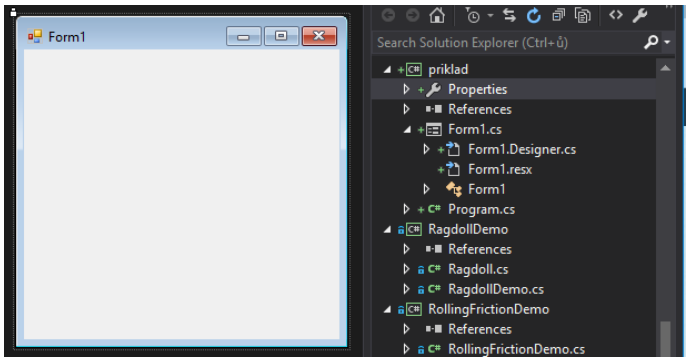
1. Do Solution Demo vytvoříme nový Windows Forms Aplication, který používá .NET framework 4.0. Pro názornou ukázku si ji pojmenujeme příklad 5.2.
2. Každý nově vytvořený framework má svoje implicitně zadané nastavení, od Visual Studia. V projektu je využíván již existující framework, proto je smazána složka Form1.cs. Ve složce Propertis - platform target je nastavena platforma x86
3. V dalším kroku je přepsáno v Program.cs jeho funkce Main(), ve které je vytvořena třída Demo , do které se nahraje třída Program. Tuto funkci následně spustíme.

```
1    using (Demo demo = new Program())
2            {
3                GraphicsLibraryManager.Run(demo);
4            }
```

4. Teď již stačí popsat funkci vytvořené scény a přidat programu reference na užívané knihovny. Jak vypadá příklad v kódu vytvoření prázdné scény, je v příloze B.3. Do složky References následně přidáme knihovny BullerSharp.dll a DemoFramework.dll.

[2] http://www.bulletphysics.org/Bullet/BulletFull/classbtCollisionWorld.html

5. Už máme vytvořenou scénu a framework, do kterého se to bude vykreslovat. Jelikož však třída Demo pro vykreslení objektu potřebuje renderer, je třeba ještě do složky Debug nahrát knihovny, které dělají rendering za nás, což jsou například knihovny SharpDX.Direct3D11, sharpdx_direct3d11_1_effects_x86 a další.
6. Teď už máme hotový spustitelný soubor, který nám sice nevykreslí nic do scény, ale nainicializuje nám všechny prvky potřebné pro další práci.



Obr. 5.2: Příklad tvorby nového Frameworku

5.2 Popis jednotlivých metod laboratorního modelu

Pro tuto práci byl vybrán laboratorní model kmeny, který má i svou reálnou podobu v laboratoři, kde se na tomto modelu učí studenti programovat PLC boxy viz. příloha C.3. Tento model se skládá z několika na sebe nezávislých částí a to pás(belt), písty a optické snímače. Jak se vytvoří scéna je již popsáno výše. V následujících kapitolách bude vysvětleno, jak se do scény definuje model Kmenu.

5.2.1 Definice globálních proměnných

V kapitole 5.1.1 bylo popsáno, jak si vytvořit třídu, která bude bude dědit metody od třídy Demo a tudíž umožní s nimi pracovat. Vytvořená třída se jmenuje Stepa_Kmeny. Tato třída kromě několika metod, které si popíšeme níže, má také několik globálních proměnných, viz přílohaC.4. Jako globální proměnná byla vytvořena pouze ta, která se použila ve více než jedné metodě.

Proměnné eye a target říkají, odkud kam se bude kamera dívat. Dále se nainicializuje pole raycastBar, ve kterém každý prvek nastavuje jeden Ray Test.

Poslední a velice důležitou skupinou globálních proměnných je inicializace proměnných a to d6body0, Generic6Dofball0, EngineSet, EngineVelocity0, MotorForce0, přičemž první dvě proměnné definují motor a zbylé tři hodnoty říkají, jestli bude motor spuštěn a jakou sílu a rychlost bude mít viz. 3.5

5.2.2 Initializace

Na začátku tvorby scény jsou objekty, které se vytvoří jen jednou a pak už se o ně nemusíme starat. Takovéto objekty, činnosti se vykonají v inicializačních metodách OnInitialize() a OnInitializePhysics().

OnInitialize V této části volá metoda Freelook s parametry eye a target. Ten nastaví výchozí pozici kamery a umožní pohyb pomocí kláves a myši po scéně. Metoda Graphics pojmenuje vytvořenou Winform aplikaci. Poslední metodou se nastaví vykreslování a můžeme určit, jestli chceme, aby se u pohybujících objektů zobrazily souřadnice jejich pohybu.

OnInitializePhysics Na začátku inicializace scény se nejdříve nastaví parametry celého světa, jeho kolize, směr a velikost gravitační síly.
Následně se vytvoří plocha, na kterou se bude celý model vykreslovat a vytvoří se statická scéna celého modelu, jenž vypadá stejně, jako vyparsovaný model C.1. Zde se také zavolá funkce pás (Belt) a písty Enginepress0 Enginepress1. Zde se také inicializuje Ray test, který je zadán pomocí dvou pozic ve scéně. Jak jednotlivé metody fungují, je popsáno níže.

5.2.3 Pás

Pohyblivý pás je tvořen pomocí několika rotujících válců, které mají stonásobně větší mass než ostatní objekty, aby při kolizi s jiným objektem nebyl válec vychýlený z jejich aktuální pozice.

V této funkci je vytvořeno 320 válců, které jsou zavěšeny pomocí kloubu Hinge a každému z nich je nastaven motor, jenž válcem otáčí. Silou targetVelocity -20.0f a silou motoru maxMotorImpulse 20.0f.

U normálního pásového dopravníku by stačilo, aby rozměry válce byly dvakrát menší než velikost tělesa, které po pásu jede. Jelikož však podstava vytvořených válců je ve skutečnosti realizována jako šestiúhelník, což neumožňuje plynulý chod objektů po válcích. Pro plynulé posunování objektů po poli válců, bylo vytvořeno větší množství malých válců.

```
1 hinge[i] = new HingeConstraint(body0[i],
2             pivotInA, axisInA);
3 hinge[i].EnableAngularMotor(true, targetVelocity,
```

```
4 |            maxMotorImpulse);
```

5.2.4 Písty

V této práci jsou vytvořeny dva identické písty, které se liší pouze svojí pozicí a ovládáním. Proto si popíšeme práci jenom jednoho z nich a to pístu popsaného funkcí Enginepress0().

Jak tato funkce Enginepress0() funguje:

1. Na začátku vytvoříme RigidBody.

```
1 | d6body0 = LocalCreateRigidBody(mass, trans, shape);
```

2. Na tuhé těleso jsem použil omezení pohybu 3.3 a to tak, že byla zakázána rotace kolem jakékoli osy a posun byl umožněn jen ve směru osy x a to v rozsahu hodnot od 0 do 10.
3. Následně je na těleso d6body0 pomoci uzlu hinge připevněn píst pipe0, kterému je omezen pohyb v jakékoli směru. Těleso pipe0 se tedy pohybuje zároveň s tělesem d6body0. Pokud však bude na kloub vyvinuta větší síla, může se toto omezení překročit.
4. V posledním kroku definice pistu se nastaví síly, které budou na vytvořený píst působit.

```
1 | EngineSet0 = true;
2 | EngineVelocity0 = new Vector3(10.0f, 0.0f, 0);
3 | MotorForce0 = new Vector3(5.0f, 0.0f, 0);
```

Jelikož však tyto hodnoty postupně měníme v metodě OnHandleInput(), jenž reaguje na stisk klávesy, musíme realizaci motoru volat v metodě OnUpdate(), která se volá stále dokola než program skončí.

```
1 | Generic6Dofball0.TranslationalLimitMotor.
2 | EnableMotor[0] = EngineSet0;
3 | Generic6Dofball0.TranslationalLimitMotor.
4 | TargetVelocity = EngineVelocity0;
5 | Generic6Dofball0.TranslationalLimitMotor.
6 | MaxMotorForce = MotorForce0;
```

5.2.5 Raytest

Pro testování kolize objektu s paprskem byla vytvořena třída RaycastBar, která zajišťuje vytvoření paprsku a zjištění kolize s objektem. Pokud kolize nastane, vypíše

se do konzole, který paprsek je v kolizi. Jak vypadá mapa kódu tříd RaycastBar, je ukázáno v přílozeC.5.

Tato třída má tři metody:

Konstruktor Paprsek se volá v již dříve zmiňované metodě třídy Stepa_Kmeny OnInitializePhysics, kde se vytvoří pomoci tří hodnot. A to dvou pozic, které udávají, odkud kam bude paprsek vykreslen a čísla, o jaké číslo testu se bude jednat.

```
1 RaycastBar(Vector3 start_position, Vector3
2  end_position, int testnumber);
```

Testování kolize Testování kolize se provádí v metodě třídy Stepa_Kmeny OnUpdate(), která se provádí stále dokola.

```
1 ClosestRayResultCallback(ref _source[i],
2 ref _destination[i]);
```

Jestliže dojde ke kolizi, zapíše se do proměnné rayHasHit hodnota true a na konzolu se vypíše, jaký paprsek je v kolizi. Následně se přepíše pozice cílu raytestu na pozici kolize.

Vykreslení paprsku Tato metoda se volá stejně, jako metoda pod metodou testování kolize. Pokud kolize nastane a přepíše se cílová pozice, musí se paprsek vykreslit jen do vzdálenosti kolize.

Tato metoda tedy řeší dva druhy vykreslení paprsku, a to od zadané pozice start_position do end_position a ze start_position do pozice hitPoint.

5.3 Zhodnocení

V této části projektu byl vytvořen komplexní laboratorní model obsahující všechny základní metody, které lze na tuhé těleso v bullet physic použít.

Model se dá ovládat pomoci kláves J, K, L, což se volá ve třídě OnHandleInput, přičemž klávesa J vytvoří největší, K střední a L nejmenší kvádr. Klávesy I a O umožňují pohyb s písty, pokud jsou zmáčknuty, píst se vysouvá. Pokud ne, písty se vrací zpět.

Pro detekci velikosti může posloužit třída RaycastBar, ve které vytvoříme raytest. Ten, pokud je v kolizi, s jakýmkoli objektem, vrátí číslo testu, který je v kolizi. Toho se dá využít pro detekci velikosti kmenu.

Obrázek, jak vypadá reálný model a laboratorní model, je v přílohách C.2 a C.3.

6 TESTOVÁNÍ DEMONSTRAČNÍHO MODELU

Posledním úkolem práce bylo otestování demonstračního modelu, to jsem provedl pomoci vyčítání FPS(počet snímků za sekundu) po dobu 25s od spuštění programu. Demonstrační model kmeny jenž je popsaný v kapitole 5. Test probýhal na notebooku Lenovo IdeaPad G580 s procesorem Intel® Core™ i3-2328M a grafickou kartou Nvidia Geforce GT630M 2GB, který byl v průběhu zatěžován různými programy na pozadí a do aplikace byl na pás vytvořen po každých pěti stech vykreslených frejmech vytvořil malý kvádr.

Tento test byl prováděn pětkrát pokaždé s během jiných program na pozadí.

Test1 V prvním testu byly počítač zatížen programy Mozzila firefox, TEX maker a Visual studio 2015. Při této zátěži dokázal program vykreslovat v průměrně 123,4 frejmů za sekundu.

Test2 V druhém testu byly počítač zatížen spuštěným Full HD filmem. Při této zátěži dokázal program vykreslovat v průměrně 55 frejmů za sekundu.

Test3 Ve třetím testu běžela jen samotná aplikace. Při této zátěži dokázal program vykreslovat v průměrně 124,2 frejmů za sekundu.

Test4 Ve čtvrtém testu byly počítač zatížen počítačovou hrou COD4. Při této zátěži dokázal program vykreslovat v průměrně 110,2 frejmů za sekundu.

Test5 V poseldním testu byl na pozadí spuštěn program Mozzila firefox s spuštěnými videi na youtube. Při této zátěži dokázal program vykreslovat v průměrně 99,3 frejmů za sekundu.

Tabulka změřených hodnot je přiložena v příloze D.1 a tyto data jsou zobrazeny v grafu C.6.

7 ZÁVĚR

Výsledkem této práce je simulátor laboratorního modelu kmeny, který se ovládá pomocí kláves, myši a X3D parses, který vybere data do paměti a následně vykreslí statickou scénu.

Na začátku byly popsány základy renderování objektů ve scéně. Jak vykreslování scény funguje, bylo popsáno na nejzákladnější pipeline tvořené z pixel shaderu a vertex shaderu. Byly zde také vysvětleny základní transformace, které lze následně s jednotlivými objekty ve scéně nebo ze scénou samotnou provádět.

V praktické části projektu byl vytvořen parser X3D formátu v jazyce C#, který dokáže z X3D formátu vybrat všechny druhy primitivních těles a následně je zobrazit ve scéně. O rendrování scény vytvořené do Windows formuláře se starala třída DemoFramework, která je volně stažitelná z internetu. K vyrendrování objektů do scény jsou použity knihovny DirectX 11, DirectX 10, DirectX 9 nebo OpenGL, přičemž záleží na uživateli, jakou knihovnu bude chtít používat. V této práci byla zvolena knihovna OpenGL.

Jako druhá aplikace byl vytvořen laboratorní model kmenů, který byl ovládán pomocí kláves a myši. Díky těmto událostem které jsou navázány na klávesnici a myš, je možné se v aplikaci pohybovat po scéně, vytvářet v ní tři druhy boxů a pohybovat s písty. Tato část projektu byla také otestována v poslední části projektu, kde bylo zjištěno, že aplikace dokáže na testovaném počítači vykreslit nejvíce 130,1 FPS, což není nijak závratná hodnota. Jelikož dnešní počítačové hry dokáží na dobrých PC vykreslit až 300 FPS, člověk však tento rozdíl hodnot nepozná, jelikož lidské oko vnímá rendrování rychlejší než 60 FPS jako plynulé. Nejnižší hodnoty byly změřeny, když na pozadí počítače běžel full HD film, který dokázal aplikaci zpomalit až na hodnotu 49,9 FPS, zde už bylo viditelné, že se objekty po scéně nepohybují plynule.

LITERATURA

[1] Renderování. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2017-01-04]. Dostupné z: https://cs.wikipedia.org/wiki/Renderování

[2] Rendering. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2017-01-04]. Dostupné z: https://en.wikipedia.org/wiki/Rendering_(computer_graphics)

[3] Pixel Shaders. In: *Nvidia* [online]. [cit. 2017-01-04]. Dostupné z: http://www.nvidia.com/object/feature_pixelshader.html

[4] Shader Stages. In: *Windows Dev Centrum* [online]. [cit. 2017-01-04]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/bb205146(v=vs.85).aspx

[5] Graphics Pipeline. In: *Windows Dev Center* [online]. [cit. 2017-01-04]. Dostupné z:https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882(v=vs.85).aspx

[6] TIŠNOVSKÝ, Pavel. Grafické karty a grafické akcelerátory (21). In: *Root.cz* [online]. Root.cz, 2005 [cit. 2017-01-04]. Dostupné z: https://www.root.cz/clanky/graficke-karty-a-graficke-akceleratory-21/

[7] TIŠNOVSKÝ, Pavel. XML + 3D = X3D. In: *Root.cz* [online]. 2008 [cit. 2017-01-09]. Dostupné z: https://www.root.cz/clanky/xml-3d-x3d/?ic=serial-box&icc=text-title

[8] VRML. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2014 [cit. 2017-01-09]. Dostupné z: https://cs.wikipedia.org/wiki/VRML

[9] Extensible Markup Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2017-01-09]. Dostupné z: https://cs.wikipedia.org/wiki/Extensible_Markup_Language

[10] X3D. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-01-09]. Dostupné z: https://en.wikipedia.org/wiki/X3D

[11] Rendering Pipeline Overview: Pipeline. In: *www.khronos.org* [online]. 2017 [cit. 2017-05-25]. Dostupné z: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

[12] Coordinate Systems*Coordinate Systems* [online]. In: . Learn OpenGl, 2014 [cit. 2017-05-25]. Dostupné z: https://learnopengl.com/#!Getting-started/Coordinate-Systems

[13] Rendering a cube. In: *Http://glumpy.readthedocs.io/en/latest/index.html#* [online]. https://readthedocs.org/, 2011 [cit. 2017-05-25]. Dostupné z: http://glumpy.readthedocs.io/en/latest/tutorial/cube-ugly.html

[14] Transformations. In: *Https://learnopengl.com* [online]. Patreon, 2014 [cit. 2017-05-25]. Dostupné z: https://learnopengl.com/#!Getting-started/Transformations

[15] BLAŽEK, Michal a Zdeněk LEHOCKÝ. XML pro začátečníky - 1. část. In: *Http://programujte.com* [online]. 2007 [cit. 2017-05-25]. Dostupné z: http://programujte.com/clanek/2007030501-xml-pro-zacatecniky-1-cast/

[16] BLAŽEK, Michal a Zdeněk LEHOCKÝ. XML pro začátečníky - 2. část. In: *Http://programujte.com* [online]. 2007 [cit. 2017-05-25]. Dostupné z: http://programujte.com/clanek/2007062201-xml-pro-zacatecniky-2-cast/

[17] Bullet 2.83 Physics SDK Manual. In: *Http://www.bulletphysics.org* [online]. Doxigen, 2015 [cit. 2017-05-25]. Dostupné z: http://www.bulletphysics.org/Bullet/BulletFull/index.html

[18] COUMANS, Erwin. *Bullet 2.83 Physics SDK Manual* [Pdf]. 2015 [cit. 2017-05-25]. ISBN -. Dostupné z: http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page

[19] X3D Example Archives. In: *Http://www.web3d.org* [online]. [cit. 2017-05-25]. Dostupné z: http://www.web3d.org/x3d/content/examples/Basic/RigidBodyPhysics/

[20] MILET, Tomáš. *Uvod do OpenGL* [online]. Božetěchova 1/2. 612 66 Brno - Královo Pole, 2016 [cit. 2017-05-25]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cwk.php?id=11496&csid=633217. Přednáška. Fakulta informačních technologií Vysokého učení technického v Brně.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

GPU	Grafický procesor – graphic processing unit
CPU	Centrální procesorová jednotka – central processing unit
OpenGL	Grafické rozhraní – Open Graphics Library
.NET Framework	Nejrozšířenější platforma pro osobní počítače
X3D	Forma na ukládání 3D scén – Extensible 3D
VRML	Grafický formát – Virtual Reality Modeling Language
XML	Značkovací jazyk – Extensible Markup Language
3D	trojrozměrný – trojdimenzionální
2D	dvourozměrný – dvoudimenzionální
FPS	počet snímků za sekundu – Frames Per Second

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

- `/` ..... kořenový adresář přiloženého CD
  - `bakalarska_prace_Svab.pdf` ..... Elektronická verze písemné části práce
  - `zdrojove_kody.zip` ..... Zdrojové kódy Simulátoru laboratorního modelu
    - `DemoFramework` ..... Nastavení využívaného .NET Frameworku
    - `Packages` ..... Balíčky knihoven
    - `xsvabs` ..... Vytvoření prázdné scény
    - `xsvabs_kmeny_BeltCylinder` ..... Simulátor laboratorního modelu
    - `xsvabs_x3d_parser` ..... X3D parser
    - `BulletSharp.dll` ..... Knihovna bullet physic
    - `CodeMap_Demoframework.dgml` ..... Mapa kódu Demoframeworku
    - `CodeMap_parse.dgml` ..... Mapa kódu X3D parseru
    - `CodeMap_Simulator_kmeny.dgml` ..... Mapa kódu simulátoru kmeny
    - `DemoFramework.dll` ..... Knihovna DemoFramework
    - `Demos.sln` ..... Solution celého projektu

B UKÁZKY KÓDU

B.1 Část X3D modelu

Výpis B.1: Část X3D modelu.

```
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN"
"http://www.web3d.org/specifications/x3d-3.3.dtd">

<X3D  id="boxes" showStat="false" showLog="false"
x="0px" y="0px" width="500px" height="500px">
  <Scene>
    <Viewpoint position="70 40 60"
    orientation="15 5 5"></Viewpoint>
    <Group>
      <CollidableShape DEF='elementShape1'>
        <Shape containerField='shape'>
          <Box size='7.5 1 5'/>
        </Shape>
      </CollidableShape>
      <CollidableShape DEF='elementShape2'>
        <Shape containerField='shape'>
          <Box size='7 1 7.5'/>
        </Shape>
      </CollidableShape>
      .
      .

      <CollidableShape DEF='elementShape8'>
        <Shape containerField='shape'>
          <Box size='1.5 2 5'/>
        </Shape>
      </CollidableShape>
      <CollidableShape DEF='belt'>
        <Shape containerField='shape'>
          <Box size='30 0.7 5'/>
        </Shape>
      </CollidableShape>
    </Group>
```

```
34      <RigidBodyCollection>
35        <RigidBody DEF='element' RotationZ='PI / 4'
36         Translation='-12.9 19.4 0' mass='0'>
37          <CollidableShape USE='elementShape1'/>
38        </RigidBody>
39        <RigidBody DEF='element1' RotationX='PI / 4'
40          Translation='0 11.3 4.7' mass='0'>
41          <CollidableShape USE='elementShape2'/>
42        </RigidBody>
43       .
44       .
45
46     <RigidBody DEF='element20' Translation='12.0 10 0'
47        mass='0'>
48          <CollidableShape USE='belt'/>
49        </RigidBody>
50
51      </RigidBodyCollection>
52    </Scene>
53  </X3D>
```

B.2 Ukázka Vrml kódu

Výpis B.2: Ukázka Vrml formátu.

```
1  #VRML V1.0 ascii
2  Separator {
3      DirectionalLight {  # nastavení %osvětlení
4          direction 0 0 -1
5      }
6      PerspectiveCamera { # nastavení %kamery
7          position        -8.6 2.1 5.6
8          orientation     -0.1352 -0.9831 %-0.1233 1.1417
9          focalDistance 10.84
10     }
11     Separator { # červená koule
12         Material {
13             diffuseColor 1 0 0
```

```
14          }
15          Translation {
16              translation 3 0 1
17          }
18          Sphere {
19              radius 2.3
20          }
21      }
22      Separator {# zelená krychle
23          Material {
24              diffuseColor 0 0 1
25          }
26          Transform {
27              translation -2.4 .2 1
28              rotation 0 1 1.9
29          }
30          Cube {}
31      }
32  }
```

B.3 Ukázka vytvoření prázdné scény

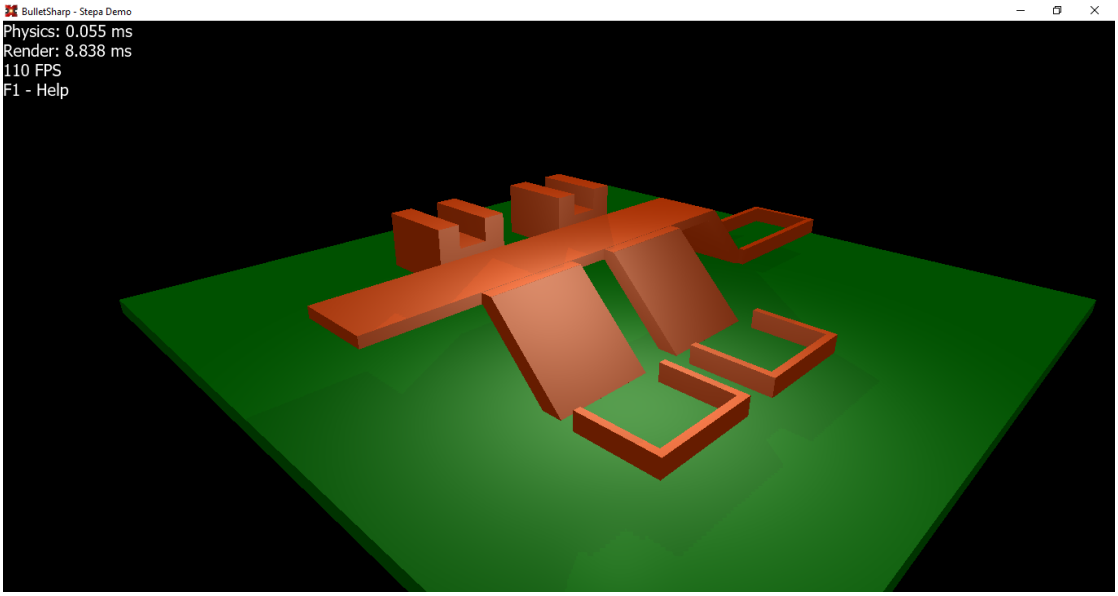
Výpis B.3: Ukázka vytvoření prázdné scény pomoci C#.

```
1  using System;
2  using BulletSharp;
3  using DemoFramework;
4  using System.Windows.Forms;
5
6  namespace Stepa_Demo
7  {
8      class Stepa_Demo : Demo
9      {
10         Vector3 eye = new Vector3(10, 30, 50);
11         Vector3 target = new Vector3(0, 5, -4);
12
13         protected override void OnInitialize()
14         {
```

```
15              Freelook.SetEyeTarget(eye, target);
16              Graphics.SetFormText("BulletSharp -
17              Stepa Demo");
18              IsDebugDrawEnabled = true;
19          }
20          protected override void OnInitializePhysics()
21          {
22              CollisionConf = new
23              DefaultCollisionConfiguration();
24              Dispatcher = new
25              CollisionDispatcher(CollisionConf);
26              Broadphase = new DbvtBroadphase();
27              Solver = new
28              SequentialImpulseConstraintSolver();
29              World = new DiscreteDynamicsWorld
30              (Dispatcher, Broadphase, Solver,
31              CollisionConf);
32              World.Gravity = new Vector3(0, -10, 0);
33          }
34      }
35      static class Program_Stepa
36      {
37          /// <summary>
38          /// The main entry point for the application.
39          /// </summary>
40          [STAThread]
41          static void Main()
42          {
43              using (Demo demo = new Stepa_Demo())
44              {
45                  GraphicsLibraryManager.Run(demo);
46              }
47          }
48      }
49  }
```

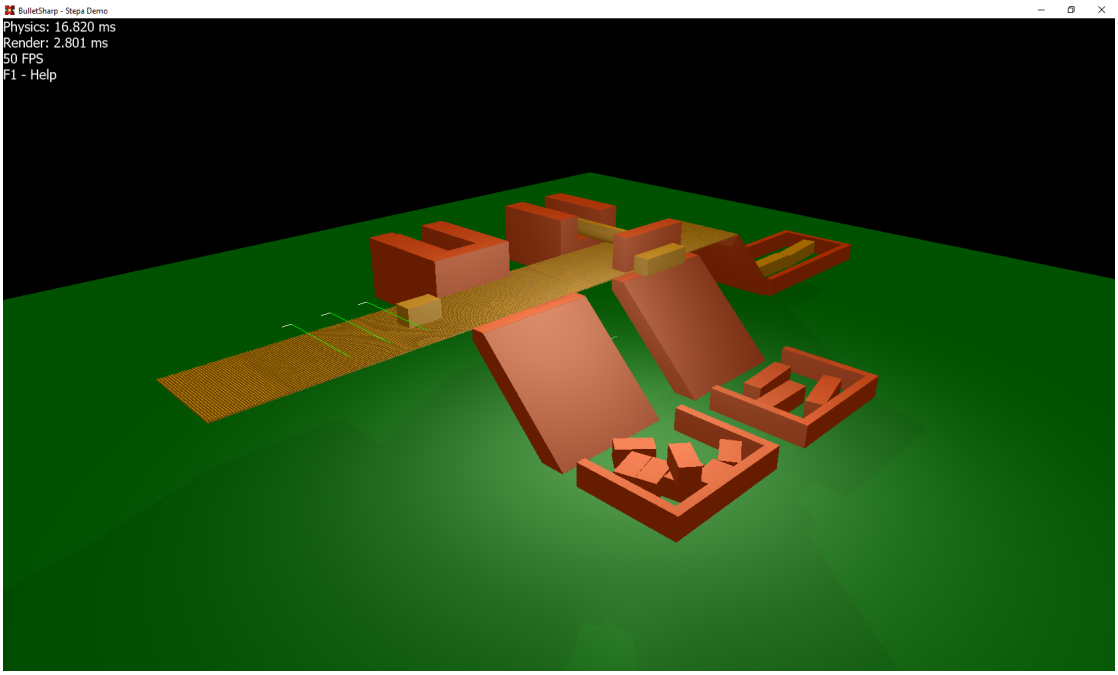
C OBRÁZKY

C.1 Print Screen X3D parseru



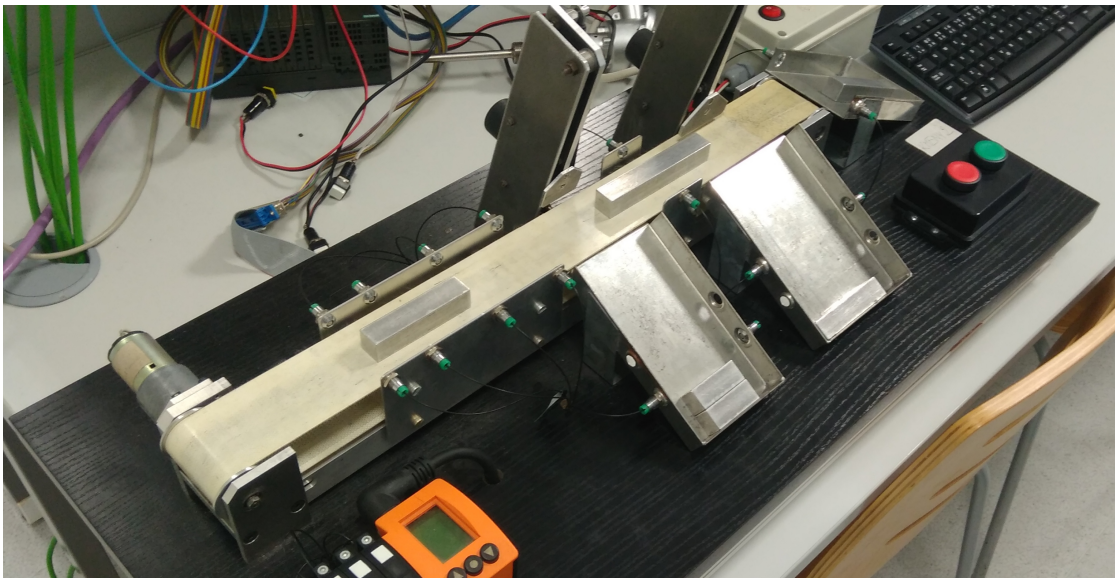
Obr. C.1: Print Screen vyrendrované scény X3D parseru

C.2 Print screen laboratorního modelu



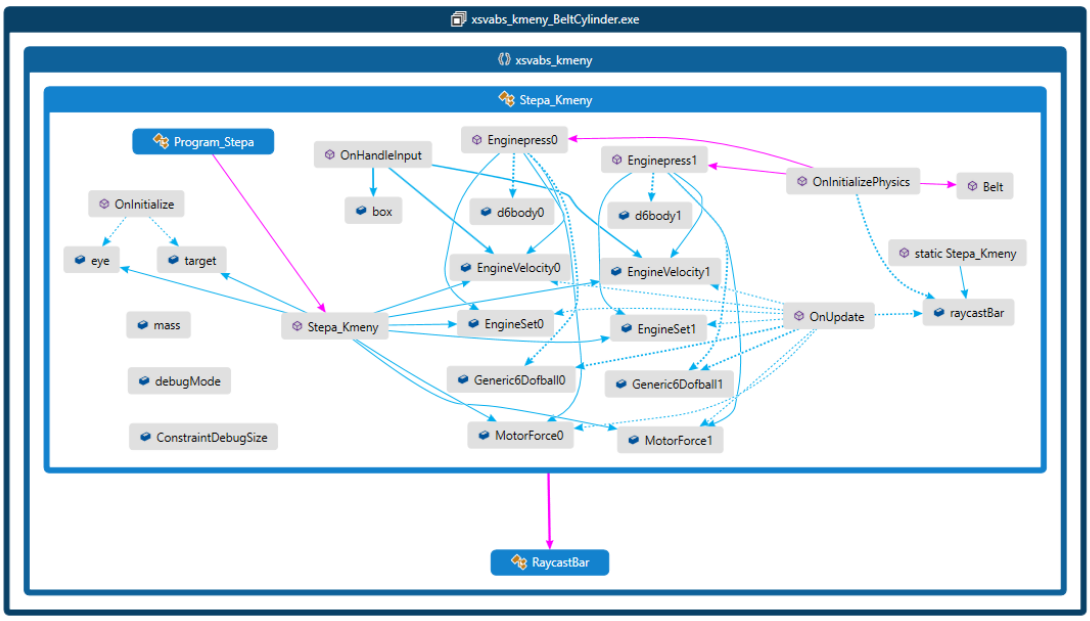
Obr. C.2: Print Screen vyrendrované scény Fyzikálního modelu

C.3 Reálný model



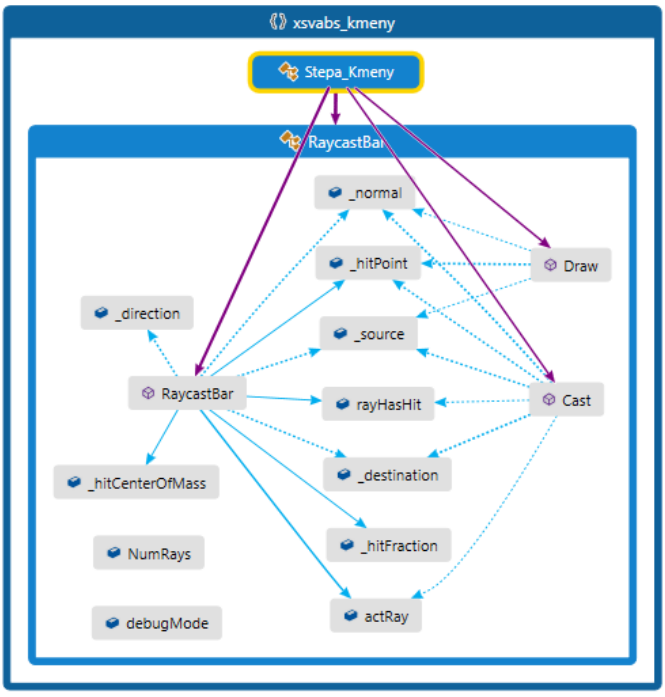
Obr. C.3: Foto reálného modelu

C.4 Mapa kódu třídy Stepa_Kmeny



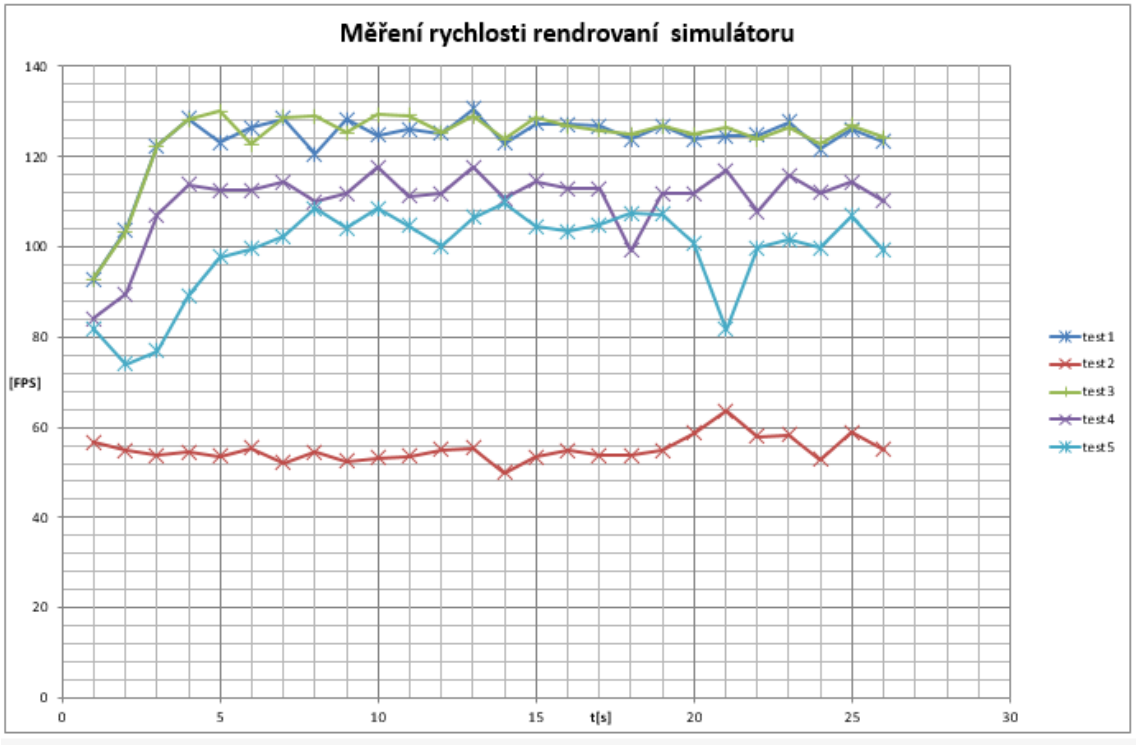
Obr. C.4: Mapa kódu třídy Stepa_Kmeny

C.5 Mapa kódu tříd RaycastBar



Obr. C.5: Mapa kódu tříd RaycastBar

C.6 Graf měření FPS



Obr. C.6: Graf měření FPS

D TABULKA MĚŘENÍ FPS

Tab. D.1: Tabulka měření FPS

Čas	**test1**	**test2**	**test3**	**test4**	**test**
[s]	[**FPS**]	[**FPS**]	[**FPS**]	[**FPS**]	[**FPS**]
1	92,7	56,6	92,9	84,0	81,8
2	103,7	54,9	103,4	89,7	74,0
3	122,4	53,7	122,3	107,0	77,0
4	128,5	54,6	128,2	113,8	89,2
5	123,1	53,5	130,1	112,6	97,7
6	126,5	55,3	122,9	112,6	99,5
7	128,5	52,2	128,8	114,3	102,3
8	120,6	54,7	128,9	110,0	108,5
9	128,2	52,5	125,5	111,9	104,2
10	124,8	53,2	129,3	117,7	108,6
11	126,2	53,5	129,1	111,3	104,5
12	125,0	55,0	125,4	111,8	100,1
13	130,7	55,3	129,0	117,7	106,4
14	123,0	49,9	123,9	110,7	109,8
15	127,4	53,3	128,8	114,6	104,5
16	127,1	54,9	126,8	112,9	103,5
17	126,7	53,8	125,9	112,8	104,8
18	123,8	53,8	124,9	99,2	107,4
19	126,7	54,8	126,7	111,9	107,3
20	123,9	58,6	124,9	111,9	100,9
21	124,5	63,6	126,3	116,9	81,7
22	124,8	57,9	123,8	107,7	99,7
23	127,6	58,3	126,5	116,0	101,7
24	121,6	52,8	123,0	112,0	99,9
25	126,1	58,8	127,0	114,4	106,9
Průměr	123,4	55,0	124,2	110,2	99,3